Programación
PL/I

Autor: CLEFormación S.L
Localidad y año de impresión: Madrid, 2010

Introducción a PLI

Descripción de los datos

Entrada / Salida

Estructura del programa

Manipulación de datos

1

Notaciones del lenguaje

TABLA DE CONTENIDOS

ANTECEDENTES

El PL/l es un lenguaje de programación de finalidades múltiples, de alto nivel, que permite programar tanto aplicaciones científicas y comerciales como aplicaciones de sistemas y de tiempo real. Permite también al programador utilizar de manera eficiente toda la potencia de su ordenador y programar aplicaciones con cierta relativa independencia de la máquina. Debido a su diseño modular, el PL/I proporciona medios a todos los programadores, cualquiera que sea su nivel de experiencia. Los principiantes solo necesitan aprender aquellas características del lenguaje que satisfagan sus necesidades, mientras que el programador experimentado tiene oportunidad de utilizar las características más avanzadas, aprovechándose de los aspectos más útiles del lenguaje a fin de poder programar más eficientemente y con mayor facilidad que en otros lenguajes de programación.

La presente publicación describe aquellas características del PL/I aplicables al proceso comercial de datos, sin limitarse a un aspecto determinado de dicho proceso, sino intentando abordar la totalidad de las características del PL/I, que pueden utilizarse en toda la gama de aplicaciones comerciales.

La naturaleza del proceso comercial de datos ha alcanzado en los últimos años su mayor grado de complejidad. Anteriormente. según crecían en volumen y complejidad las operaciones de negocios los problemas inherentes al proceso comercial de datos se caracterizaban por sus soluciones fragmentarias. tales como incremento del personal implicado en los problemas o mayor utilización de los equipos. Pero ya en los últimos años ha llegado a evidenciarse la necesidad de emplear métodos de sistemas totales, planificaciones formales de largo alcance y técnicas científicas para resolver los problemas del proceso comercial de datos, hasta cierto punto de forma similar a como se habían necesitado tales técnicas para resolver los problemas de producción. Consecuentemente, el campo de actuación del programador comercial ha aumentado.

Además de ser responsable de actividades tales como el diseño de informes, la producci6n de nóminas y la resolución de problemas contables el programador comercial está implicándose de modo creciente en problemas más complicados, tales como toma de decisiones, programación lineal y previsiones de producción. Esto significa que el programador comercial se relaciona ocasionalmente con los aspectos científicos del proceso de datos; puede requerirse de él que procese datos binarios y realice cálculos aritméticos en formato dé punto flotante. En la presente publicación se estudian tales consideraciones, aunque sin desarrollarlas detalladamente, el propósito estriba en familiarizar al programador comercial con aquella características del PL/I utilizables en aplicaciones comerciales avanzadas. Esta publicación consta de cinco capítulos y un apéndice. El capítulo primero estudia las notaciones del lenguaje, versando sobre ternas tales como el juego de caracteres del lenguaje y las reglas de formación de identificadores. El capítulo 2 describe los diversos tipos de datos procesados por los programas en PL/I. El capitulo 3 contiene un estudio sobre entrada y salida. En el capítulo 4 se presentan la estructura de los programas en PL/I y el control de ejecución de las sentencias. En el capítulo se desarrollan el tratamiento y compaginación de los datos.

Dado el gran número de analogías del PL/I con el COBOL (Common Business Oriented Language), cada capítulo de esta publicación concluye con una sección que muestra comparaciones entre ambos lenguajes, comparaciones que no son necesariamente parte de este manual pudiendo pasarlas por alto quienes carezcan de conocimientos de COBOL.

INTRODUCCIÓN

Los lenguajes de programación pueden clasificarse en lenguajes orientados al ordenador y lenguajes orientados a problemas. Las instrucciones que se emplean en programas escritos en lenguajes orientados al ordenador se especifican en notaciones que reflejan la composición de instrucciones de máquina.

Palabras, tales como ADD siglas como TSX; campos como los de dirección, pueden estar contenidos en las instrucciones de dichos programas. La cantidad de letras que cabe utilizar en una palabra, el conjunto de siglas y el orden de los campos dentro de una instrucción se fijan de acuerdo con el orden y tamaño de los campos en una instrucción de máquina. En resumen, un lenguaje orientado al ordenador se diseña para un ordenador determinado, y no cabe aplicarlo en uno de distinto tipo.

En un lenguaje orientado a problemas la notación refleja el tipo de problema que se trata de resolver, antes que el tipo de ordenador donde se va a procesar el programa. En COBOL las notaciones empleadas para escribir un programa se parecen al inglés; las notaciones en FORTRAN son parecidas al lenguaje de las matemáticas; las notaciones del PL/I combinan las características de COBOL y las notaciones de FQRTRAN.

Así como existen restricciones en las notaciones del inglés y de las matemáticas, existen también en la notación de los lenguajes orientados a problemas. Al escribir un programa sólo puede utilizarse un juego específico de números, letras y caracteres especiales, debiendo observarse reglas especiales de puntuación y uso de blancos.

El resto de este capitulo es un estudio de las reglas de notación que se emplean al escribir programas con PL/I.

FORMATO DE PL/1

Permite al programador escribir un programa en formato libre.

Dependiendo de la configuración, puede establecerse ciertas normas, de modo que se puedan preparar programas para un ordenador por medio de registros de longitud física, como los que se preparas en ISPF, de 80 caracteres.

Escribiremos programas entre las columnas 1 a 72 (o 2 a 72, dependiendo de las opciones del compilador).

PALABRAS, DELIMITADORES, BLANCOS Y COMENTARIOS

Todos los elementos que componen un problema en PL/I se construyen a partir del juego de caracteres del PL/l. Hay dos excepciones: las constantes de series de caracteres y los comentarios, los cuales pueden contener cualquier carácter permitido para una configuración de máquina determinada.

Los programas enl PL/I constan de palabras y/o delimitadores. Las palabras pertenecen a una de estas dos categorías: identificadores o constantes. Las palabras adyacentes pueden estar separadas por uno o más delimitadores y/o blancos. Por ejemplo,

- a CALLA se le considera como un identificador, y a CALL A cómo dos identificadores;
- a AB+BC se le considera como dos identificadotes separados por el delimitador +, siendo equivalente a AB + BC donde el signo + está precedido y seguido de blancos.

Puede emplearse comentarios en cualquier lugar donde los blancos estén permitidos, excepto dentro de una constante de series de caracteres o una especificación de modelo. Los comentarios tienen la forma:

```
/*comentario*/
```

y pueden constar de uno o más de los caracteres permitidos para una determinada configuración de máquina. Sin embargo, no puede figurar dentro de un comentario la combinación de caracteres */, puesto que significa la terminación de tal comentario.

JUEGO DE CARACTERES Y DELIMITADORES

El juego de, caracteres del PL/I comprende 60 caracteres. Estos son caracteres alfabéticos del idioma Inglés, dígitos decimales y caracteres especiales.

Hay 29 caracteres denominados alfabéticos que son las letras A a Z, el símbolo monetario (escrito $), el signo comercial de arrobas (escrito @) y el signo sostenido (escrito #).

Existen 10 dígitos, 0 a 9, y 21 "caracteres especiales". Los nombres y los signos gráficos por los que se representan son:

SEPARADORES	SIGNO	UTILIZACION
	,	separa los elementos de una lista
	.	separa los calificadores de nombres
	;	finaliza sentencias
	:	sigue a los símbolos de sentencia y a los prefijos de condici6n
	.	separa los calificadores de nombres
	()	se utiliza en expresiones y, para encerrar listas y especificar inrormación asociada a ciertas palabras clave.
	%	Indica sentencias que han de ser procesadas por el precompilador
OPERADORES ARITMETICOS	SIGNO	UTILIZACION
	+	denota suma o cantidad positiva
	-	denota resta o cantidad negativa
	*	denota multiplicación
	**	denota exponenciación
	/	denota división

OPERADORES DE COMPARACION	SIGNO	UTILIZACION
	>	Mayor que
	¬>	No mayor que
	>=	Mayor o igual a
	=	Asignación y comparación lógica
	<=	Menor o igual a
	<	Menor que
	¬<	No menor que
OPERADORES LOGICOS	SIGNO	UTILIZACION
	¬	Denota negación
	&	Denota simultaneidad (y)
	|	Denota disyuntiva (o)
OPERADOR CONCATENACION	SIGNO	UTILIZACIÓN
	||	Concatenar literales
	_	Para identificadores

IDENTIFICADORES

Un identificador puede ser una palabra creada por el usuario para identificar un archivo, una partida o ítem de datos, o alguna o todas las partes de su programa; o bien, puede ser una de las palabras utilizadas para identificar elementos del lenguaje PL/I, tales como sentencias, atributos y opciones. Este último tipo de identificador se denomina palabra clave.

La palabra DECLARE sería un ejemplo de palabra clave cuando se escriba como primera palabra de una sentencia DECLARE, puesto que, al ser la primera palabra identifica a tal sentencia como una DECLARE.

Todos los identificadores, se usen o no como palabras clave, deben escribirse de acuerdo con las reglas siguientes:

- El identificador puede estar formato por caracteres alfabéticos, dígitos y el subrayado, debiendo empezar por un carácter alfabético.
- Dentro del identificador se permite cualquier número de subrayados (_). Incluso consecutivos.
- Los identificadores deben estar formados por un máximo de 31 caracteres.

SENTENCIAS.

Las palabras, delimitadores, blancos y comentarios estudiados hasta ahora, se utilizan en PL/I para formar sentencias, que son los elementos básicos para construir los programas de PL/I. Se emplean para describir los datos, para el proceso real de los datos y para controlar la secuencia de ejecución de otras sentencias. Todas las sentencias en PL/I terminan en un punto y coma. Con excepción de la sentencia de Asignación, que se estudia en el Capítulo 5, y la sentencia de Invalidez estudiada en el Capítulo 4, todas las sentencias en PL/I deben empezar por una palabra clave.

COMPARACION ENTRE PL/I Y COBOL:' *NOTACIONES DEL LENGUAJE*

En el estudio que sigue se comparan las notaciones de los lenguajes PL/I y COBOL. En general, ambos emplean notaciones similares: las palabras definidas por el programador emplean caracteres alfabéticos y dígitos decimales; las expresiones constan de sucesiones de nombres, constantes y operadores; las palabras clave identifican los elementos del lenguaje; los elementos están separados por caracteres de puntuación; las sentencias tienen apariencia similar al inglés.

No obstante las reglas de notación de ambos lenguajes difieren de hecho; en la lista que sigue figuran algunas de las diferencias más significativas.

- Tanto en PL/I como en COBOL, las palabras clave tienen significados atribuidos de antemano. En COBOL, la utilización de las palabras clave se limita al fin a que se destinan, no pudiéndose utilizar para otros fines. En PL/I J sin embargo, las palabras clave no se reservan para fines especiales, pudiendo aparecer en cualquier lugar donde esté permitida una palabra definida por el programador; por ejemplo, la palabra clave READ cabe utilizarla como nombre de un fichero en un programa en PL/I. Los diferentes significados de una misma palabra se determinan en PL/I, según el contenido del texto, lo cual permite crear palabras clave para nuevas características del lenguaje, evitando también tener que reprogramar programas fuente antiguos, en que las palabras definidas por el programador pueden ser incompatibles con las nuevas palabras clave.

- El COBOL requiere un formulario de programación; el PL/I no lo requiere. Los caracteres de puntuación en PL/I determinan el significado y agrupación de los elementos del lenguaje, lo cual permite tratar los programas en PL/I como series largas de caracteres, y transmitirlos al ordenador mediante casi cualquier medio de entrada.
- En COBOL deben existir blancos antes y después de los operadores aritméticos, lo cual no es necesario en PL/I. En PL/I sólo se requieren blancos entre palabras sucesivas que no estén separadas por caracteres especiales como paréntesis, operadores y signos de puntuación.
- El COBOL restringe los comentarios a la Procedure Division, siendo necesario que dichos comentarios estén escritos en una sentencia NOTE. En PL/I se admite que aparezcan comentarios a todo lo largo del programa permitiendo utilizarlos donde puedan figurar blancos (excepto en una constante de series de caracteres o en una especificaci6n de modelo; estas características del lenguaje se estudian en el Capítulo 2).
- El juego de caracteres del COBOL consta de 51 caracteres; el PL/1 emplea 60.

Los puntos que siguen muestran algunas de las diferencias menos significativas.

- Todas las sentencias de PL/I finalizan en punto y coma; las sentencias de COBOL finalizan en punto, coma o punto y coma.
- En COBOL, las palabras definidas por el programador no deben exceder de 30 caracteres; el límite en PL/I es de 31 caracteres.
- Para mejorar la legibilidad, el PL/I utiliza el subrayado (_) en las palabras definidas por el programador; el COBOL emplea el guión medio (-).

2

Descripción de los Datos

TABLA DE CONTENIDOS

INTRODUCCIÓN

El estudio que se desarrolla a continuación, versa sobre los tipos de datos que pueden emplearse en una aplicación de programación PL/I. También se estudian las características de los distintos tipos de datos y los métodos disponibles para su organización en configuraciones, tales como conjuntos dimensionados.

No obstante, este capítulo se limita a estudiar la descripción y organización de los datos dentro de la memoria interna de los ordenadores.

La descripción de los datos tal como aparecen en los medios externos de almacenamiento, se verán en el Capítulo 3.

VARIABLES Y CONSTANTES

Un campo de datos es el valor de una variable o de una constante. Una variable tiene un valor o conjunto de valores que pueden cambiarse durante la ejecución de un programa. Una variable surge, generalmente por una declaración, que declara el nombre y ciertos atributos de la variable. Una referencia a una variable es uno de los siguientes:

- Un nombre declarado
- Una referencia sacada de un nombre declarado por:
  - puntero
  - estructura
  - índice

Un constante tiene un valor que no puede cambiarse.

Los tipos de variables o constantes son:

Tipo de variable	Atributos de datos
Aritméticos	REAL|COMPLEX FLOAT|FIXED BINARY|DECIMAL (precision)
Cadenas de caracteres	BIT|CHARACTER|GRAPHIC (length) [VARYING]
Picture	PICTURE REAL|COMPLEX

TIPOS DE DATOS

Las instrucciones que ejecuta un ordenador pueden dividirse en tres categorías generales:

- Lógicas: manejan secuencias de bits y de caracteres.
- Aritméticas: procesan datos numéricos.
- Instrucciones de Control: determinan el orden en que se ejecutan otras instrucciones.

Pueden emplearse categorías similares para clasificar las operaciones facilitadas por el PL/1. Para cada una de estas categorías de operaciones existe un tipo de dato correspondiente que es usado por ellas. En PL/1 estos tipos de datos se denominan:

- Datos de series
- Datos aritméticos
- Datos considerados como símbolos.

DATOS DE SERIES.

Se utilizan esencialmente en operaciones lógicas y constan de secuencias de caracteres o bits. Los ítems de datos de series pueden dividirse en dos categorías generales:

- Ítem de datos de series de caracteres
- Ítem de datos de series de bits.

ÍTEM DE DATOS DE SERIES DE CARACTERES

Constan de una secuencia de caracteres, que en PL1 pueden ser cualquiera que permita el ordenador.

Se usan esencialmente en operaciones como comparar, compaginar e imprimir.

```
DCL FIN_FICHERO CHAR(1) INIT('N');
DCL NOMBRE CHAR(15) INIT('TOMAS');
```

ÍTEM DE DATOS DE SERIES DE BITS.

Constan de una secuencia de bits, cada uno de los cuales representa una condición "on" u "off". La condición "on" se representa por un bit 1 y la "off" por un bit 0. En los programas de PL/I, es frecuente usar ítem de datos de series de bits en operaciones lógicas, cuyos resultados se emplean para fines de control.

```
DCL FIN_FICHERO BIT(1) INIT('0'B);
```

DATOS ARITMÉTICOS

Representan información numérica y se usan en operaciones aritméticas. Existen dos clases de ítem de datos aritméticos:

- Decimales
- Binarios.

Pudiendo venir cada uno de ellos en representación de punto fijo o punto flotante.

ÍTEM DE DATOS DECIMALES.

Representan información numérica y constan de una secuencia de dígitos decimales. Para cada dígito decimal se definen una combinación exclusiva de bits.

```
DCL CONT_REGISTROS DEC FIXED(7,2)  INIT(0);
```

ÍTEM DE DATOS BINARIOS.

Representan información numérica y constan de una secuencia de bits. Aunque los ítem de datos binarios utilizan bits, no son equivalentes a los ítem de datos de series de bits. Un ítem de datos binarios representa un valor numérico, mientras que un item de datos de series de bits representa una secuencia de condiciones "on" - "off".

```
DCL IND_TAB  BIN FIXED(15)  INIT(1);
DCL SQLCODE BIN FIXED(31)  INIT(0);
```

FORMATOS DE PUNTO FIJO Y PUNTO FLOTANTE.

Los ítem de datos decimales y binarios pueden estar en formato de punto fijo o en formato de punto flotante.

Con frecuencia el formato de punto flotante da como resultado formas más compactas que el de punto fijo. Este caso se da generalmente cuando se necesita una gran cantidad de ceros para fijar la posición del punto decimal o binario en los formatos de punto fijo.

Por ejemplo, la fracción decimal de punto fijo .000000009 necesita 8 ceros para establecer la posición del punto decimal. en formato de punto flotante no se necesitan los ceros porque la posición del punto decimal se establece mediante un exponente entero que aparece en el dato en punto flotante.

DATOS CONSIDERADOS COMO SÍMBOLOS

SÍMBOLOS DE SENTENCIAS.

En los programas en PL1, las operaciones de procesos de datos se especifican mediante sentencias. A las sentencias pueden atribuirles símbolos, de forma que sea posible referirse a ellas mediante otras sentencias, como las sentencias de control, que alteran la secuencia de ejecución de las sentencias.

```
DCL ETIQUETA LABEL INIT('PROC1');
```

En ciertas sentencias puede utilizarse un símbolo como ítem de datos (en el Capítulo 4 se estudian los ítem de datos considerados como símbolos.

NOMBRES DE ITEM DE DATOS.

Con frecuencia es necesario utilizar en los programas un nombre que identifique los ítem de datos a procesar. Dicho nombre identificativos se denomina nombre de datos. Los nombres de datos se ajustan a las reglas establecidas en el capítulo 1.

Los diferentes ítem de datos que pueden identificarse por el mismo nombre de datos deben tener las mismas características de datos, las cuales se asocian al nombre de datos, al escribirlas junto con dicho nombre en una sentencia DECLARE, que se verá en este capítulo.

CONSTANTES

En los programas PL1, no siempre es necesario asignar nombres a todos los ítems de datos. En realidad los ítem de datos pueden figurar dentro del programa, estando por tanto disponibles para su uso inmediato en el programa. Los ítems de datos que figuran en un programa en PL1 se llaman constantes.

Por cada tipo de dato que están permitidos en los programas PL/I, existe su correspondiente tipo de constante, disponible para su uso en los programas.

CONSTANTES DE SERIES DE CARACTERES.

Pueden tener cualquier carácter reconocible por un ordenador en concreto. Estas constantes se encierran entre comillas sencillas, por ejemplo:

- '$123.45'
- 'PERICO PEREZ'
- '45.62'.

Las comillas no forman parte de la constante. Si se desea representar una comilla como parte de la constante, debe utilizarse comilla doble. Por ejemplo:

- 'IT"S'.

Es posible indicar la repetición de una constante de series de caracteres anteponiendo a la especificación de la constante un entero decimal (encerrado entre paréntesis) que indique el número de repeticiones. Por ejemplo:

- (3)'*-*' que es equivalente a escribir '*-**-**-*'.

CONSTANTES DE SERIES DE BITS.

Están formadas por una secuencia de los dígito 1 - 0 encerrados entre comillas simples e inmediatamente seguidos por la letra B. Por ejemplo:

- '0100'B

Es posible especificar la repetición de una constante de series de bits de forma similar a como se hace para las constantes de series de caracteres:

- (10)'1'B, que es equivalente a '111111111'B.

CONSTANTES DE DECIMALES DE PUNTO FIJO

Se compone de uno o más dígitos (de 0 a 9), pudiendo tener punto decimal. Por ejemplo:

- 72.192
- .567
- 2010

Como se observa no se encierran entre comillas.

CONSTANTES DE BINARIAS DE PUNTO FIJO

Se compone de uno o más dígitos 1 o 0, pudiendo tener punto binario y seguido de la letra B:

- 110011B
- 11.1100B
- .0011B

Y tampoco se encierran entre comillas.

CONSTANTES DE DECIMALES DE PUNTO FLOTANTE

Se compone de una constante decimal de punto fijo seguida de la letra E, la cual a su vez, va seguida de un exponente con signo optativo. El exponente es un entero decimal y representa una potencia de 10, que determina el emplazamiento real del punto decimal.

Por ejemplo, la constante decimal de punto flotante 123.45E+5 tiene el mismo valor que la expresión aritmética $123.45+10^5$; esta expresión es igual en valor a la constante decimal de punto fijo 12345000.

Otros ejemplos de constantes decimales de coma flotante son:

- 420.3E-14
- 45.E-4

CONSTANTES DE BINARIAS DE PUNTO FLOTANTE

Se compone de una constante binaria de punto fijo seguida de la letra E, la cual a su vez, va seguida de un exponente con signo optativo; el exponente le sigue inmediatamente la letra B. El exponente es un entero decimal y representa una potencia de dos que determina la situación real del punto binario.

Por ejemplo la constante binaria de punto flotante .101E+9B, es equivalente al valor de la constante binaria de punto fijo 101000000B.

Otros ejemplos:

- 1.1011E-3B
- 1111.E+20B
- 10101E5B

CONSTANTES CONSIDERADAS COMO SÍMBOLOS

Identifica una sentencia y se ajusta a las reglas de formación de identificadores establecidas en el capítulo1. En PL1 se asignan símbolos a las sentencias escribiendo una constante inmediatamente a la izquierda de la sentencia, la cual sucede por tanto a la constante.

Un ejemplo de constante considerada como símbolo es la palabra MENSAJE en la siguiente sentencia:

- MENSAJE: DISPLAY('FIN DE TRABAJO');

Mensaje es la constante de símbolo que identifica a la sentencia Display.

El uso de las constantes consideradas como símbolos se estudia en el capítulo 4.

LA SENTENCIA DECLARE

En los programas PL1, las descripciones explícitas de las características de los datos se escriben en forma de sentencias.

La sentencia DECLARE se usa para describir datos con nombres tal como dichos datos están representados en la memoria interna del ordenador.

Estas propiedades que caracterizan a un Ítem de datos se llaman Atributos.. Los atributos de un Ítem de datos con nombre se especifican mediante palabras clave, las cuales pueden aparecer junto con el nombre de un Item de datos en una sentencia Declare.

La forma general de una sentencia declare se puede escribir:

- DECLARE nombre_1 atributos,.....,nombre_n atributos;

Los atributos se separan por blancos. El último atributo que figura con un nombre va seguido de una coma, excepto al final de la sentencia Declare, que lleva punto y coma.

Los atributos para los datos se dividen en tres categorías, cada una de las cuales corresponden a uno de los tres tipos de datos generales que se estudiaron previamente:

- Atributos aritméticos.
- Atributos de series
- Atributos de símbolos.

ATRIBUTOS ARITMÉTICOS

El PL1 proporciona atributos aritméticos para describir ítems de datos que tienen un valor numérico, y son:

- DECIMAL, BINARY, FIXED y FLOAT

atributo de precisión, especifica el número de dígitos asignados a ese ítem de datos. Para datos de punto fijo, este atributo especifica también la posición del punto decimal supuesto.

El atributo de precisión consta, bien de un solo entero decimal encerrado entre paréntesis, bien de dos enteros decimales, separados por una coma y encerrados entre paréntesis (8,3). En la sentencia DECLARE, el atributo de precisión debe ir inmediatamente precedido por uno de los atributos: DECIMAL, BINARY, FIXED O FLOAT. Sólo pueden intercalarse blancos.

Para campos de punto flotante, solo se escribe en el atributo de precisión un entero, que indica el número de dígitos que aparecen en el campo antes de la E.

Para campos de punto fijo, se escriben normalmente dos enteros; el primero indica el número de dígitos que figuran en el campo de datos, y el segundo, el número de dígitos a a la derecha del punto decimal o binario. Si no va a haber ningún dígito a la derecha del punto decimal, solo es necesario escribir el primer numero.

Considérese la siguiente declaración:

```
DECLARE    SALARIO DECIMAL FIXED (7,2),
           CALCULO FLOAT DECIMAL (10),
           CONTADOR FIXED (5) BINARY,
           MEDIO BINARY (10) FLOAT;
```

El campo asignado a salario sería decimal de punto fijo, compuesto por 7 dígitos, suponiendo que dos de ellos están a la derecha del punto decimal.

El campo asignado a CALCULO sería decimal de punto flotante, con diez dígitos antes de la E (ej.: .1234567891E5).

El campo asignado a CONTADOR sería binario de punto fijo, compuesto de cinco dígitos, suponiendo que ninguno de ellos está a la derecha del punto decimal.

ATRIBUTOS DE SERIES

Especifican los ítem de datos de series de caracteres o bien los de series de bits. Son CHARACTER y BIT.

ATRIBUTO CHARACTER

Especifica que el nombre de datos de la sentencia DECLARE representa series de caracteres. Y se escribe de la forma:

```
CHARACTER (longitud).
```

La especificación de longitud es una constante entera decimal que indica el numero de caracteres de los ítem de datos.

cuando la serie de caracteres es de longitud variable, se escribe lo siguiente:

```
CHARACTER (longitud) VARYING
```

La especificación de longitud indica el número máximo de caracteres de una serie de caracteres de longitud variable.

```
DECLARE    CABECERA CHARACTER(80),
           TITULO CHARACTER(40) VARYING;
```

Especifica que la variable de series de caracteres denominada CABECERA, es de longitud fija y consta de 80 caracteres, y que el número de caracteres del campo TITULO puede variar de 0 a 40.

ATRIBUTO BIT

Especifica que el campo nominado en la sentencia DECLARE se representa por una serie de Bits. El atributo Bit puede aparecer con dos formatos:

```
BIT (longitud)
BIT (longitud) VARYING
```

La especificación de longitud y el atributo VARYING tienen el mismo significado que el descrito para el atributo Characer, excepto que la longitud indica el número de bits del ítem de datos de series de bits.

ATRIBUTO PICTURE

Se usa para especificar las características detalladas de los ítem de datos de series y, frecuentemente para compaginar el formato de estos campos que van a imprimirse. La compaginación de estos campos puede implicar la sustitución de ciertos caracteres, tales como los ceros no significativos, por otros caracteres, y la inserción, dentro de la serie, de caracteres especiales de moneda, o el punto.

```
PICTURE 'especificación del modelo'
```

Tal como se indica, la especificación del modelo debe encerrarse entre comillas sencillas. Consta de una secuencia de caracteres denominados caracteres de modelo.

En el estudio que sigue solo se trata aquellos caracteres de modelo que permiten que el atributo PICTURE sirva como alternativa a los aritméticos y los de series.

Estos caracteres son: A, X, 9, V, l, K, S. Los restantes caracteres de modelo se usan para compaginación de datos (Capítulo 5).

Aunque el atributo PICTURE pueda servir como alternativa para los atributos aritméticos, y, por tanto especificar la representación de un valor numérico, el campo que se describe por medio del atributo PICTURE es un ítem de series de caracteres, y no de datos aritméticos. Por tanto, la eficacia de las operaciones del ordenador puede resultar afectada al efectuar cálculos aritméticos en ítem de datos descritos con el atributo PICTURE. En la mayoría de los ordenadores, esta falta de eficiencia proviene de la imposibilidad de efectuar cálculos aritméticos directamente en los ítems de series de caracteres. Por tanto es necesario convertir su representación de series de caracteres a representación aritmética (ver Capítulo 5, conversión de datos aritméticos).

ATRIBUTOS DECIMALES DE PUNTO FIJO CON PICTURE

Para especificar un ítem de datos de series de caracteres con atributos decimales de punto fijo, pueden utilizarse los caracteres 9 y V del modelo. En la especificación del modelo, una secuencia de caracteres 9 indica que las posiciones de caracteres correspondientes en el ítem de datos de series de caracteres contienen siempre dígitos decimales (0 a 9). El carácter V especifica que deberá suponerse el punto decimal en la posición correspondiente del campo de series de caracteres. El V supone punto decimal, pero internamente en memoria no está representado. El carácter V solo puede aparecer una vez en el modelo.

Ejemplos:

```
DECLARE    Cuenta PICTURE '999',
           Coste PICTURE '999V99',
           Tarifa PICTURE 'V999';
```

ATRIBUTOS DECIMALES DE PUNTO FLOTANTE CON PICTURE

Puede especificarse los atributos decimales de punto flotante para un ítem de datos de series de caracteres usando los caracteres del modelo: 9, V, K, S.

La especificación del modelo para los atributos decimales de punto flotante consta de dos partes:

- La primera contiene los caracteres del modelo para un ítem de datos decimales de punto fijo.
- La segunda, situada inmediatamente a la derecha del primero, representa el exponente del punto flotante.

La especificación del modelo para un exponente decimal de punto flotante empieza en la letra K, seguida del carácter optativo S, después del cual aparece una secuencia de caracteres 9. La letra K no representa un carácter real en el ítem de datos de series de caracteres, sino que especifica la posición inicial del exponente en la serie de caracteres. El carácter S indica que figura un signo + o - en la posición correspondiente del campo de series de caracteres, y especifica el signo del exponente.

```
DECLARE media PICTURE 'V999KS99'
```

ATRIBUTOS SERIES DE CARACTERES CON PICTURE

Para especificar ítem de datos de series de caracteres, se usan los caracteres de modelo X y A. En la especificación del modelo, el carácter X indica que la posición de carácter correspondiente puede contener cualquier carácter posible.

El carácter de modelo A se emplean para representar letras y el carácter blanco.

```
DECLARE    nombre PICTURE 'AAAAA',
           simbolo PICTURE 'XXXXXXXXXX',
           codigo PICTURE 'AAXXX';
```

REPETICIÓN DE CARACTERES EN PICTURE

Las sucesivas apariciones del mismo carácter en una especificación del modelo puede indicarse colocando un entero decimal entre paréntesis delante del carácter que va a repetirse. El valor del entero decimal especifica el número de repeticiones. Por ejemplo la sentencia:

```
DECLARE    bruto PICTURE '(7)9V99',
           Parte PICTURE '(6)A(5)X(2)9',
           Fraccion PICTURE 'V(8)9';
```

Es equivalente a:

```
DECLARE    bruto PICTURE '9999999V99',
           Parte PICTURE 'AAAAAAXXXXX99',
           Fraccion PICTURE 'V99999999';
```

EXCLUSIVIDAD DE ATRIBUTOS

A los atributos descritos anteriormente se les aplican las reglas siguientes:

- BIT, CHARACTER y PICTURE no deben emplearse para describir un mismo ítem de datos.
- Un ítem descrito con BIT, CHARACTER o PICTURE no debe describirse también con FIXED, FLOAT, BINARY o DECIMAL.

EL ATRIBUTO DEFINED

Con frecuencia conviene poderse referir al mismo ítem de datos mediante nombres diferentes, particularmente cuando en el mismo programa intervienen varios programadores y cada uno emplea un nombre diferente para el mismo ítem de datos. Para este propósito puede utilizarse el atributo DEFINED, que se escribe de la forma siguiente:

```
atributos del nuevo-nombre DEFINED nombre-antiguo
```

Considérese la siguiente sentencia:

```
DECLARE TITULO   CHARACTER(80),
        CABECERA CHARACTER (80) DEFINED TITULO;
```

Esta sentencia especifica que el ítem de datos identificado por TITULO es un ítem de datos de series de caracteres, en número de 80, y que el mismo ítem de datos puede identificarse por el nombre CABECERA. Obsérvese que para el nuevo nombre se especifican tos atributos del ítem, incluso siendo idénticos a los atributos especificados para el nombre antiguo.

Cabe utilizar también el atributo DEFINED con ítems de datos aritméticos como se ilustra en la siguiente sentencia:

```
DECLARE  SALARIO FIXED DECIMAL (5,2),
         ABONO   FIXED DECIMAL (5,2) DEFINED SALARIO;
```

Esta sentencia especifica que SALARIO es el nombre de un ítem de datos aritméticos y que es posible dirigirse al mismo ítem de datos mediante el nombre ABONO.

En ítems de datos de series, también es posible utilizar el atributo DEFINED para aplicar un nombre de datos a una parte de una serie. Esto se consigue mediante el empleo de un atributo de series de caracteres en el nuevo nombre para especificar el número de caracteres de la serie a los que se aplica el nuevo nombre. Por ejemplo, la sentencia siguiente:

```
DECLARE FECHA CHARACTER(6),
         MES  CHARACTER(2) DEFINED FECHA;
```

especifica que FECHA es el nombre de un ítem de datos de series de caracteres en número de seis, y que los dos primeros caracteres de dicha serie se denominan MES.

Cuando se aplica el nombre nuevo a una parte del ítem de datos de serie que no empieza con el primer carácter de una serie de caracteres o con el primer bit de una serie de bits el atributo DEFINED se escribe junto con el atributo siguiente:

```
POSITION (constante-entera-decimal)
```

La constante entera decimal del atributo POSITION especifica la posición inicial de la parte del ítem de datos de series a la cual se refiere el nuevo nombre. Cuando la posición inicial es la primera posición de la serie, no se requiere el atributo POSITION. Considérese la siguiente sentencia:

```
DECLARE FECHA CHARACTER(6),
        MES   CHARACTER(2) DEFINED FECHA,
        DIA   CHARACTER(2) DEFINED FECHA POSITION (3),
        AÑO   CHARACTER(2) DEFINED FECHA POSITION (5);
```

Esta sentencia especifica que el ítem de datos de series de caracteres denominado FECHA consta de seis caracteres, y también que los dos primeros caracteres de la serie FECHA constituyen la serie de caracteres denominada MES, que el tercer y cuarto caracteres constituyen otra serie denominada DIA y que el quinto y sexto caracteres forman otra serie denominada AÑO.

Los atributos que aparezcan con el nuevo nombre en el atributo DEFINED deben ser congruentes con los atributos declarados para el nombre antiguo; por ejemplo, no puede emplearse el atributo BIT para el nombre nuevo si para el antiguo se empleó el atributo CHARACTER. Más aún, el atributo POSITION tiene que especificar la posición inicial de acuerdo con la longitud indicada por los atributos del nombre nuevo. Considérese la siguiente sentencia:

```
DECLARE  PARTE  CHARACTER(10),
         MODELO CHARACTER(4) DEFINED PARTE  POSITION(8);
```

Esta sentencia contiene una incongruencia. El atributo POSITION(8) indica que el nombre MODELO se refiere a una serie de caracteres que consta de los caracteres octavos noveno, décimo y undécimo de la serie denominada PARTE. Sin embargo, la serie de caracteres denominada PARTE consta de 10 caracteres, no de 11; por tanto, la sentencia anterior es incorrecta.

ATRIBUTOS POR OMISIÓN

En PL/I no es necesario que exista una descripción explícita de cada nombre del programa en una sentencia DECLARE. Las características de algunos nombres de datos se sobreentienden en el contexto en que dichos nombres aparecen, y no necesitan detallarse. Las características de otros nombres de datos pueden describirse solo parcialmente; cabe especificar ciertos atributos para estos nombres de datos descritos parcialmente y omitir otros atributos.

Cuando no se describe explícitamente un nombre de datos, o cuando se describe parcialmente al nombre de datos se le suponen aplicados ciertos atributos que se llaman "atributos por omisión". En tal caso, se dan por supuestas las siguientes hipótesis de omisión:

- Cuando no se ha dado ninguna descripción explícita del nombre de datos, si el nombre empieza por una letra I a N se supone que tiene los atributos FIXED BINARY, mientras que si el nombre empieza por otra letra distinta de I a N, se supone que tiene los atributos FLOAT DECIMAL.
- Si un nombre tiene el atributo BINARY o DECIMAL. se supone que también tiene el atributo FLOAT. Salvo especificación en contrario.
- Si un nombre tiene el atributo FIXED o FLOAT, se supone que tiene también el atributo DECIMAL, salvo especificación en contrario.
- La especificación de precisión por omisión se definirá separadamente para cada aplicación de PL/I.

CONFIGURACIONES DE DATOS

En PL/I, los ítems de datos pueden agruparse para formar configuraciones de datos. El lenguaje proporciona dos clases de configuraciones:

- estructuras y
- conjuntos dimensionados.

Es posible referirse a las configuraciones o a partes de estas configuraciones mediante un solo nombre.

ESTRUCTURAS

Los ítems de datos que ni tienen idéntico tamaño ni son del mismo tipo, pero que poseen entre sí cierta relación lógica, pueden agruparse en jerarquías llamadas estructuras. El concepto general de estructura se evidencia en otros muchos campos, además del de proceso de datos.

La organización de un libro ilustra una aplicación del concepto de estructura. El libro puede dividirse en diversas partes, y cada parte puede constar de uno o más capítulos. Los capítulos pueden comprender varios temas principales, y cada tema puede estar formado por subtemas, etc. Normalmente esta jerarquía se muestra en una tabla de materias. Utilizando sangrados y diferentes tipos de letras de forma que de una ojeada resulte evidente la organización del libro.

Los datos de un programa en PL/I pueden organizarse en jerarquías muy parecidas a las de un libro. Considérese una estructura de datos conteniendo un ítem que es una dirección. A la dirección puede dársele el nombre de datos DIRECCJON. Sin embargo, si el programador tiene que referirse a las partes individuales de la dirección, debe nombrarlas. Por ejemplo, CALLE, CIUDAD, ZONA y ESTADO. En este caso, DIRECCION es de nivel superior a CALLE, CIUDAD, ZONA Y ESTADO, e incluye cada una de ellas. Una, vez especificada la subdivisión de un ítem de datos, puede subdividirse aún más para permitir referencias más detalladas. Las subdivisiones más elementales de una estructura de datos, es decir, aquéllas que no se subdividen más, se llaman ítems elementales de datos.

Para especificar la organización de tos ítem elementales de datos en estructuras y a su vez la organización de las estructuras en estructuras más amplias, se emplea en PL/I un sistema de números de nivel.

- Los números de nivel empiezan en 1 para las estructuras principales, es decir, estructuras no contenidas en otras estructuras.
- A las estructuras menos amplias, es decir, las inferiores se les asignan los números de nivel más altos (no necesariamente sucesivos).

En general, cuanto más alto sea el valor del número de nivel, más bajo será su nivel jerárquico.

Al describir una estructura en una sentencia DECLARE, el número de nivel asociado con un nombre de datos debe figurar inmediatamente delante de este nombre de datos. Los atributos de los ítems elementales de una estructura aparecen después del nombre con el que están asociados.

Obsérvese que los nombres de datos especificados para todos tos ítems de una estructura excepto los elementales, son nombres de niveles. Una referencia a dicho nombre en el cuerpo de un programa es, de hecho, una referencia a los ítems elementales subordinados a ese nombre.

En el ejemplo que sigue se ilustra el efecto de los niveles:

```
DECLARE
1 REGISTRO_CUENTAS_CORRIENTES,
   2 NOMBRE,
     5 PRIMER_APELLIDO   CHARACTER (15).
     5 NOMBRE_DE_PILA    CHARACTER (10),
     5 SEGUNDO_APELLIDO  CHARACTER (9),
   2 NUMERO_CUENTA   CHARACTER (9), '
   2 DIRECCION,
     4 CALLE CHARACTER (20),
     4 CIUDAD CHARACTER (15),
     4 ZONA CHARACTER (5),
     4 PAIS CHARACTER (15),
   2 SALDO FIXED (7,2);
```

Esta es la descripción de un registro de un archivo maestro que contiene las cuentas corrientes de un banco.

Cada registro contiene información concerniente a una cuenta corriente. En este ejemplo, a NOMBRE se le asigna el nivel 2 , e inmediatamente después figuran las tres entradas descriptivas de PRIMER_APELLIDO. NOMBRE_DE_PILA y SEGUNDO_APELLIDO, entradas que se identifican como contenidas en NOMBRE puesto que siguen a NOMBRE sin haber intervenido ningún otro. Nombre de datos con número de nivel igual o menor que el de NOMBRE, y porque dichas entradas tienen números de nivel mayores que el de NOMBRE. La DIRECCION también se le asigna el nivel 2, e inmediatamente a continuación figuran las cuatro entradas CALLE, CIUDAD, ZONA Y ESTADO, que se identifican como contenidas en DIRECCION de la misma forma que PRIMER_APELLIDO, NOMBRE_DE_PILA Y SEGUNDO_ APELLIDO se identifican como contenidas en) NOMBRE.

Considérese este otro ejemplo, que es una parte de un registro de nómina:

```
DECLARE
1 REGISTRO_ NOMINA,
  3 NUMERO_EMPLEADO CHARAC.TER(6),
  3 NOMBRE,
       27 PRIMER_APELLIDO   CHARACTER (15),
       27 NOMBRE_ DE_PILA   CHARACTER (15),
       27 SEOUNDO_APELLIDO  CHARACTBR (10),
  3 DIRECCION,
       12 CALLE CHARACTER (20),
       12 C IU DAD CHARACTER (15),
       12 ZONA CHARACTER (5),
       12'PAIS CHARACTER (15),
  3 FECHA_CONTRATO,
        4 MES   FIXED (2),
        4 DIA   FIXED (2),
        4 AÑO,
           14 DECADA FIXED (1),
           14 AÑO FIXED (1),
  3 TARIFA_DE_PAGO FIXED (7,2);
```

En este ejemplo se ilustran las principales reglas para asignar números de nivel, que pueden resumirse corno sigue:

- El nivel 1 se reserva para identificar a las estructuras principales.
- Un nivel con el nombre B está contenido en un nivel con el nombre A si se dan todas las circunstancias siguientes:
  - B sigue a A.
  - B tiene asignado un número de nivel más alto que el asignado a A.

  - Entre A y B no aparece ningún nombre con número de nivel igual o más bajo que el de A. Así, aunque DECADA tiene un número de nivel más alto que PAIS, no forma parte de PAIS ya que entre PAIS y DECADA aparecen nombres de datos con números de nivel inferiores a 12 (por ejemplo, FECHA_CONTRATO).
- No es necesario asignar consecutivamente los números de nivel. Así podría asignarse a MES, DIA Y AÑO cualquier número de nivel más alto que el asignado a FECHA_CONTRATO y más bajo que el asignado a DECADA.
- Una estructura puede incluir más de un nivel. Sin embargo todos los nombres de datos que componen un nivel dentro de la misma estructura (por ejemplo MES, DIA Y AÑO) deben tener el mismo número de nivel.
- Cuando un nombre de datos debe tener un número de nivel más bajo que el nombre que inmediatamente le precede, el número de nivel debe elegirse entre los de las estructuras que incluyen al nombre de datos precedente. Así, debe asignarse a DIRECCION el nivel 3, porque es el único número de nivel asignado a una estructura que contiene el nombre de datos precedente SEGUNDO_APELLIDO. Al nombre de datos DlRECCION no podría asignársele el número 1 de nivel por no ser una estructura principal. A TARIFA_DE_PAGO no podría asignársele uno de los dos números de nivel 3 o 4, ya que el nombre de datos AÑ0 está contenido en dos estructuras, una con número de nivel 4 y otra con número de nivel 3. No obstante, en caso de asignarle el número de nivel 4, se le trataría como si fuera parte de la estructura llamada FECHA_CONTRATO, que puede no ser la intención del programador.

NOMBRES CALIFICADOS

Cuando se especifican nombres de ítems elementales o de estructuras inferiores, resulta conveniente, con frecuencia utilizar el mismo nombre de datos para ítems de partes diferentes de la estructura, o utilizar el mismo nombre de datos para ítems de dos estructuras distintas.

Por ejemplo, cada un de las dos estructuras principales de nivel 1 denominadas CARDIN y CARDOUT podrían tener un ítem elemental común llamado PARTNO. Si se mencionara el ítem PARTNO en el cuerpo de un programa en PL/I la referencia no sería exclusiva, es decir, no estaría claro si PARTNO es el ítem elemental de la estructura CARDIN o el de la estructura CARDOUT. Por consiguiente. Siempre que se utilice un nombre de datos no exclusivo, debe asociarse, con uno o más nombres de la estructura que lo contiene, a fin de hacer exclusiva la referencia al mismo.

Hacer exclusivo un nombre se llama en PL/I, calificación.

EI nombre de un Ítem elemental o de una estructura inferior (no de nivel 1) se califica precediéndole del nombre de la estructura a la que pertenece dicho Ítem. Estos nombres se disponen de izquierda a derecha, en orden de número de nivel creciente, y se separan uno de otro y del nombre al que califican por puntos. Los puntos pueden ir precedidos y seguidos de blancos.

El nombre de una estructura principal no puede calificarse.

Considérense, por ejemplo, las estructuras siguientes:

```
DECLARE
1 CARDIN,  2 PARTNO, 2 DESCRIPCION,
1 CARDOUT, 2 PARTNO, 2 DESCRIPCION;
```

Se pueden mencionar los ítems elementales de datos mediante los siguientes nombres calificados:

```
CARDIN.PARTNO
CARDlN.DESCRIPCION
CARDOUT.PARTNO
CARDOUT.DESCRIPCION
```

Solo es necesario calificar un nombre mediante aquellos nombres de estructuras que hacen al nombre exclusivo. Considérese la estructura siguiente:

```
DECLARE
1 MATRIMONIO, 2 HOMBRE, 3 NOMBRE, 3 FECHA,
              2 MUJER,  3 NOMBRE, 3 FECHA;
```

Los ítems elementales de datos que tiene esta estructura pueden mencionarse de este modo:

```
HOMBRE.NOMBRE o  MATRIMONIO.HOMBRE.NOMBRE
MUJER.NOMBRE  o  MATRIMONIO.MUJER.NOMBRE
HOMBRE.FECHA  o  MATRIMONIO.HOMBRE,FECHA
MUJER.FECHA   o  MATRIMONIO.MUJER.FECHA
```

Si la estructura que sigue:

```
DECLARE
1 NACIMIENTO, 2 MUJER, 3 NOMBRE, 3 FECHA;
```

aparece en el mismo programa que la estructura anterior, los ítems elementales de datos de ambas estructuras pueden mencionarse como:

```
HOMBRE.NOMBRE                o  MATRIMONIO.HOMBRE.NOMBRE
MATRIMONIO.MUJER.NOMBRE
NACIMIENTO.NOMBRE            o  NACIMIENTO.MUJER.NOMBRE
```

Las estructuras inferiores en este caso, se mencionan como:

```
MATRIMONIO.HOMBRE
MATRIMONIO.MUJER
NACIMIENTO.MUJER
```

CONJUNTOS DIMENSIONADOS

Los ítem de datos que tienen las mismas características, es decir, son del mismo tipo de datos y del mismo tamaño (aunque no tengan necesariamente el mismo valor), pueden agruparse para formar un conjunto dimensionado.

La forma más simple de conjunto dimensionado es una secuencia de ítems de datos. Dicho conjunto sería análogo a una estructura con un solo nivel inferior. Así, una estructura que constara de 50 ítems elementales, siendo el valor de cada ítem el nombre abreviado de 1 de los 50 estados de EE.UU., podría especificarse escribiendo lo siguiente:

```
DECLARE
1 ESTADO,
    2 ALABAMA CHARACTER (4),
    •
    •
    •
    2 WYOMING CHARACTER (4);
```

Una forma más sencilla de escribir este grupo de datos consistiría en especificarlo como un conjunto dimensionado de 50 elementos debiendo tener cada uno de ellos el mismo tamaño y ser del mismo tipo de datos que los demás.

Dicho conjunto puede especificarse mediante un entero decimal entre paréntesis, que representa el número de elementos del conjunto, colocado inmediatamente detrás del nombre del conjunto dimensionado. A esta especificación le siguen después los atributos de los ítems del conjunto. De esta forma, en el ejemplo precedente, la estructura principal ESTADO y los ítems elementales subsidiarios podría especificarse como sigue:

```
1 DECLARE ESTADO(50) CHARACTER(4);
```

Considérese la siguiente descripción de estructura:

```
DECLARE
1 REGISTRO_VERIFICACION,
    2 NOMBRE,
        3 PRIMER_APELLIDO    CHARACTER (15),
        3 NOMDRE_DE_PILA     CHARACTER (15),
        3 SEGUNDO_APELLIDO   CHARACTER (15),
    2 DIRECCION,
        3 CALLE              CHARACTER (IS),
        3 CIUDAD             CHARACTER (15),
        3 PAIS               CHARACTER (15);
```

Puesto que todos los ítems elementales de REGISTRO_VERIFICACION tienen el mismo tamaño y son del mismo tipo de datos cabría especificar esta estructura como un conjunto dimensionado. Como tal conjunto, solamente constaría de ítems elementales, los cuales se dividirían en dos grupos de tres ítems cada uno.

La agrupación de ítems dentro de la estructura REGISTRO_VERIFICACION podría representarse visualmente como sigue:

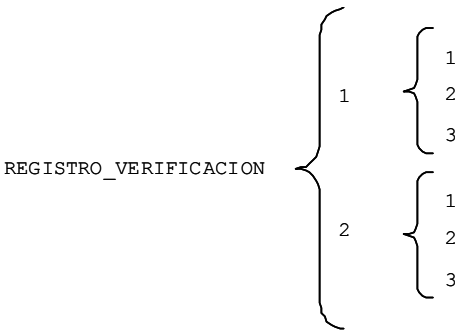
NIVEL 1	NIVEL 2	NIVEL 3
		NOMBRE_DE_FILA
	NOMBRE	PRIMER_APELLIDO
		SEGUNDO_APELLIDO
REGISTRO_VERIFICACIÓN		
		CALLE
	DIRECCION	CIUDAD
		PAIS

Dicho conjunto se especificaría en una sentencia DECLARE de la forma siguiente:

```
DECLARE REGISTRO_VERIFICACION (2,3) CHARACTER (15);
```

Los enteros entre paréntesis que siguen a REGISTRO_VERIFICACION indican la forma en que se agrupan Los ítems en este conjunto dimensionado. El primer entero (2) indica que hay dos grupos principales de ítems. El segundo entero (3) indica que cada grupo principal contiene tres ítems. Obsérvese que la estructura REGISTRO_VERIFICACION no podría haberse especificado como conjunto dimensionado si cada grupo principal hubiera tenido una cantidad distinta de Ítems dentro de él.

La agrupación de ítems dentro del conjunto dimensionado REGISTRO_VERIFICACION puede representarse visualmente como sigue:



La forma de agruparse los ítems en un conjunto dimensionado es análoga a los niveles utilizados en la especificación de una estructura. Así, en el ejemplo anterior, los grupos de tres ítems del conjunto son análogos al nivel 3 de la estructura; el grupo de dos ítems del conjunto es análogo al nivel 2; y el ítem simple REGISTRO_VERIFICACION, que es el nombre del conjunto, es análogo al nivel 1 de la estructura.

Del mismo modo que la estructura REGISTRO_VERIFICACION podría ampliarse para contener otro nivel, sería posible también ampliar el conjunto dimensionado REGISTRO_VERIFICACION para contener otro grupo. Considérese la siguiente ampliación de la estructura REGISTRO_VERIFICACION:

```
DECLARE REGISTRO_VERIFICACION (2,3,2) CHARACTER( 15);
```

Los enteros entre paréntesis que siguen a REGISTRO_VERIFICACION indican que en conjunto hay dos grupos principales que cada uno de estos grupos principales se dividen en tres grupos inferiores y que cada grupo inferior contiene dos ítems elementales (en total, doce ítems elementales. A cada conjunto de grupos se le llama dimensión.

En este ejemplo, hay un conjunto de grupos principales, un conjunto de grupos inferiores y uno de grupos elementales; es decir, una dimensión principal, una inferior y una elemental. En PL/I los conjuntos dimensionados pueden tener cualquier número de dimensiones. Así, en el ejemplo anterior, el conjunto dimensionado REGISTRO_VERIFICACION (2,3,2) podría ampliarse a más de las tres dimensiones especificadas por los enteros entre paréntesis. Para indicar el número de ítems de una dimensión pueden utilizarse expresiones en lugar de enteros.

Por ejemplo, se podría escribir:

```
'DECLARE TABLA (A+B,2,4);
```

El número de grupos de la primera dimensión sería igual al valor entero de la suma de A más B (léase las expresiones estudiadas en el Capitulo 5, "Sentencias de Asignación").

SUBINDICACIÓN

Un nombre calificado se utiliza para referirse a una estructura inferior o a un ítem elemental de una estructura. Un nombre subindicado se utiliza para referirse a un ítem de un conjunto dimensionado. Considérese un conjunto dimensionado que se especifica como sigue:

```
DECLARE REGISTRO_VERIFICACION(2,3) CHARACTER(15);
```

Con objeto de referirse al segundo ítem de este conjunto dimensionado se escribiría REGISTRO_VERIFICACION (1,2). Cada entero de la especificación denomina subíndice. El primer número se refiere al primer (principal) grupo del conjunto dimensionado; el segundo entero se refiere al segundo ítem dentro del primer grupo.

El primero de varios subíndices siempre representa uno de los grupos de la primera dimensión de un conjunto; el segundo subíndice representa uno de los grupos de la siguiente dimensión más significativa del conjunto dimensionado y así sucesivamente; el último de varios subíndices representa la posición del ítem elemental dentro de la dimensión menos significativa del conjunto dimensionado. Los subíndices se separan uno de otro por una coma. Y todos los subíndices de una referencia determinada se encierran entre paréntesis.

Por ejemplo, los Ítems del conjunto dimensionado REGISTRO_VERIFICACION y los ítems a que se refieren son:

```
REGISTRO_VERIFICACION (1,1)  al primer  ítem del primer grupo
REGISTRO_VERIFICACION (1,2)  al segundo ítem del primer grupo
REGISTRO_VERIFICACION (1,3)  al tercer  ítem del primer grupo
REGISTRO_VERIFICACION (2,1)  al primer  ítem del  segundo grupo
REGISTRO_VERIFICACION (2,2)  al segundo ítem del segundo grupo
REGISTRO_VERIFICACION (2,3)  al tercer  ítem del segundo grupo
```

No es necesario que un subíndice esté representado por una constante, sino que puede estarlo por un nombre de datos o una expresión. Así, los siguientes nombres subindicados podrían escribirse:

```
REGISTRO_VERIFICACION (A,2)
REGISTRO_VERIFICACION (A,B)
REGISTRO_VERIFICACION (A+1,2)
```

En el primero de los ejemplos anteriores, se utilizaría el valor de A como el del primer subíndice. En el segundo ejemplo se utilizaría el valor de A como el de primer subíndice y el valor de B, como el del segundo subíndice. En el tercer ejemplo, se utilizaría el valor de la expresi6n A + 1 (la suma de 1 más el valor de A) como el valor del primer subíndice.

ESTRUCTURAS QUE CONTIENEN CONJUNTOS DIMENSIONADOS.

En. PL/I, los ítems de una estructura pueden ser conjuntos dimensionados. Considérese, por ejemplo, la siguiente especificación de estructura (escrita aquí sin los atributos de los datos):

```
LISTA :DECLARE
   1 CARDIN,
      2 NOMBRE,
      2 SALARIO (2),
--',       3 ÓRDINARIO (3),
           3 EXTRAORDINARIO;
```

La especificación entre paréntesis (2) indica que la estructura SALARIO es un conjunto dimensionado que consta de dos grupos. Cada grupo contiene los ítems ORDINARIO y EXTRAORDINARIO. El ítem ORDINARIO también es un conjunto dimensionado que consta de tres Ítems. El significado de esta estructura puede describirse mediante una estructura equivalente que no contiene conjunto dimensionados:

```
DECLARE
  1 CARDIN,
      2 NOMBRE,
      2 SALARIO,
         3 0RDINARI0_1,
         3 ORDINARIO_2,
         3 ORDINARI0_3,
         3 EXTRAORDINARIO,
      2 SALARIO_2,
         3 ORDINARIO_J,
         3 ORDINARIO_2,
         3 ORDINARJO_3,
         3 EXTRAORDINARIO;
```

Se puede utilizar una combinación de clasificación y subindicacion para referirse a un ítem de un conjunto dimensionado que está contenido en una estructura. De esta forma, al objeto de referirse al segundo elemento de ORDINARIO en el primer grupo de SALARIO de la especificación de la estructura LISTA, podría escribirse:

```
CARDIN.SALARIO (1).ORDINARI0(2)
```

En el ejemplo precedente, los subíndices están escritos después de los nombres de estructura a los que se aplican. Esta es la forma normal de escribir combinaciones de calificación y subindicación. En tanto que el orden de los subíndices permanezca inalterado, dichos subíndices pueden trasladarse a la derecha de los nombres a los que normalmente se aplican.

Por ejemplo, el anterior nombre calificado y subindicado podría escribirse:

```
CARDIN.SALARIO.0RDINARIO (1 ,2)
```

ATRIBUTOS ALIGNED Y PACKED

Los atributos ALIGNED y PACKED solamente se utilizan para estructuras o conjuntos dimensionados formados por datos de serie. Estos atributos especifican la disposición en la memoria principal de todos los elementos de series de caracteres o de series de bits que componen un determinado conjunto dimensionado o estructura. El atributo PACKED indica que no debe existir memoria principal sin utilizar entre dos elementos cualesquiera de series de bits o de series de caracteres de una estructura. El atributo ALIGNED indica que los elementos de un conjunto dimensionado o estructura van a empezar en un determinado límite de la memoria principal Y, consecuentemente, puede haber memoria sin utilizar entre los elementos. El atributo ALIGNED produce con frecuencia el resultado de un proceso más eficiente, especialmente cuando se requiere un gran volumen de operaciones de empaquetado y desempaquetado. (El límite de memoria se define individualmente para cada aplicación.)

Cuando se especifican estos atributos, se aplican a todos los elementos de una estructura principal o de un conjunto dimensionado. Si se escriben para una estructura, deben escribirse para el nombre de datos con número de nivel 1 y si se escriben para un conjunto dimensionado dicho conjunto no puede ser parte de una estructura.

Estos atributos pueden escribirse en una sentencia DECLARE como se indica:

```
DECLARE
1 ETRUCTURA PACKED,
     2 SUBESTR_1 BlT (4),
     2 SUBESTR_2  BIT(5);
DECLARE CONJUNTO (4,2,4) ALIGNED;
```

ATRIBUTO LIKE

BI atributo LIKE especifica que el nombre que se está declarando es una estructura con una subestructura, cuyos elementos tienen atributos nombre idénticos a los nombres y atributos de los elementos de la estructura que se denomina. El atributo tiene el siguiente formato

```
LIKE  nombre-estructura
```

Considérense, por ejemplo, las estructuras principales A y x, descritas con la sentencia DECLARE siguiente:

```
DECLARE
1 A,
   2  CAMPO 1,
        3 DTL1  PICTURE 'AA',
        3 DTL2  CHARACTER (10),
   2  CAMPO2   CHARACTER (12),
1 X,
   2  CAMPO1,
        3 SUBCAMPO LIKE A.CAMPO1,
   2  CAMPO2  LIKE A.CAMPO1,
   2  CAMPO3   CHARACTER (5);
```

Esta especificación de la estructura principal x es equivalente a escribir:

```
1X,
  2CAMPO1,
    3 suBCAMPO1,
        4 DTL1 PICTURE 'AA',
        4 DTL2 CHARACTER (l0),
  2 CAMPO2,
    3 DTL1 PICTURE 'AA',
    3 DTL2 CHARACTER (10),
  2 CAMPO3 CHARACTER (5);
```

Es de observar lo siguiente:

- Se mantiene la diferencia entre el número de nivel del nombre de la estructura que sigue a LIKE y los números de nivel de los ítems subordinados al nombre de la estructura.
- Los nombres y atributos de los ítems subordinados al nombre de la estructura se subordinan al ítem con atributo LIKE.
- Los números de nivel de cualesquiera ítems que sigan al de atributo LIKE deben ser menores o iguales al número de nivel del Ítem con el atributo LIKE.
- Él nombre de estructura que sigue a LIKE puede estar calificado, pero no subindicado.
- Ni el nombre de estructura que sigue a LIKE, ni cualesquiera ítems subordinados al mismo pueden describirse con el atributo LIKE.

ATRIBUTO LABEL

Se puede utilizar la palabra clave LABEL para especificar que un nombre es un nombre simbólico que representa un valor, el cual es una constante considerada como símbolo. Por ejemplo, la sentencia:

```
DECLARE INDICADOR LABEL;
```

especifica que el valor del nombre simbólico INDICADOR es una constante considerada como símbolo. Un nombre simbólico puede también identificar un conjunto dimensionado de constantes consideradas como símbolos. Considérese la sentencia siguiente:

```
DECLARE SWITCH (10) LABEL;
```

Esta sentencia especifica que SWITCH es el nombre de un conjunto dimensionado de 10 elementos, y que cada elemento del conjunto es una constante considerada como símbolo. El empleo de los nombres simbólicos se estudia en el Capítulo 4.

ATRIBUTO INITIAL

El atributo INITIAL especifica las constantes atribuidas a los nombres de datos cuando se asigna la memoria del ordenador (véase "Asignación de Memoria en el Capítulo 4). El Atributo INITIAL puede escribirse de la forma siguiente:

```
INITIAL (constante)
```

Las constantes que aparecen en el atributo INITIAL deben regirse por las reglas establecidas en páginas anteriores de este capítulo para la escritura de constantes. Considérese la sentencia siguiente:

```
DECLARE MAXIMO FIXED DECIMAL (4) INITIAL (1500);
```

Esta sentencia especifica que el ítem de datos identificado por MAXIMO es un entero decimal de punto fijo que consta de cuatro dígitos y que el ítem de datos asignado MAXIMO cuando se asigna la memoria del ordenador es equivalente a la constante 1500.

En la sentencia siguiente:

```
DECLARE NOMBRE CHARACTER (4) INITIAL ('JUAN'),
        SWITCH LABEL INITIAL (DEDUCCIONES);
```

el ítem de dalos inicial de series de caracteres asignado NOMBRE es la constante 'JUAN' y la constante inicial asignada al nombre simbólico SWITCH es la constante de símbolo DEDUCCIONES.

También puede utilizarse el atributo INITIAL para asignar constantes a conjuntos dimensionados. Cuando se usa con tales conjuntos, el atributo INITIAL puede escribirse de la siguiente forma:

```
INITIAL (constante_1, …..,constante_n)
```

Las constantes se asignan en orden sucesivo a las sucesivas posiciones de un conjunto dimensionado. Considérese la sentencia siguiente:

```
DECLARE PRECIOS (4) FIXED DECIMAL (4,2) INITIAL (10.99, 20.49, 75.39, 99.99),
INDICADOR (3) LABEL INITIAL (EXTRAORDINARIAS, COMISION, PRIMA);
```

En esta sentencia se declara que PRECIOS es un conjunto unidimensional que contiene cuatro ítems de datos decimales de punto fijo. Los ítems de datos iniciales asignados al conjunto dimensionado PRECIO s se especifican mediante las constantes: 10.99, 20.49, 75.39 Y 99.99. El conjunto unidimensional denominado INDICADOR contiene tres símbolos que inicialmente son especificados por las constantes consideradas como símbolos: ExTRAORDINARIAS) COMISION y PRIMA.

EXPRESIÓN DE ATRIBUTOS COMO FACTOR COMÚN

En la sentencia DECLARE pueden asociarse, mediante el método conocido con el nombre de expresión como factor común uno o más atributos con un conjunto de nombres. La expresión como factor común se lleva a cabo agrupando entre paréntesis un conjunto de nombres delante del atributo o atributos asociados. Para separar los miembros de cado conjunto, se utilizan comas. Por ejemplo, la sentencia;

```
DECLARE (AÑO, MES, DIA) CHARACTER (2);
```

asocia el atributo CHARACTER de longitud 2 con los ítems denominados AÑO, MES Y DIA. Los atributos que se aplican a un solo miembro de un conjunto expresado como factor común se escriben siguiendo a dicho miembro. Por ejemplo:

```
DECLARE (AÑO INITIAL (62) MES, DIA) CHARACTER (2);
```

A su vez, es posible expresar como factor común un conjunto de ítems expresados de esta forma y sus atributos asociados. Por ejemplo:

```
DECLARE (AÑO, MES, DIA) CHARACTER (2),
        (HORA. MINUTO)  FIXED (2)) INITIAL (00);
```

Los números de nivel pueden expresarse corno factor común. En este caso los nombres asociados con el número de nivel, figuran entre paréntesis, siguiendo al número de nivel. Por ejemplo:

```
DECLARE
1 REGISTRO,
    2(NOMBRE,3(NOMBRE_DE_PILA, SEGUNDO_ APELLIDO, PRIMER_APELLIDO),
      EDAD, SALARIO) CHARACTER (12);
```

COMPARACIÓN ENTRE PL/I Y COBOL: DESCRIPCIÓN DE LOS DATOS

El estudio que sigue compara los medios de descripción de datos en PL/I y COBOL. En general, ambos lenguajes utilizan métodos similares para describir las características de los ítems de datos almacenados en el ordenador:

Para denominar los ítems de datos se utilizan palabras definidas por el programador; las palabras clave especifican las características de los ítems almacenados; los ítems de datos pueden agruparse en configuraciones; para cada tipo de datos pueden especificarse constantes; se pueden atribuir , valores iniciales a los nombres de datos y las especificaciones de modelo pueden servir como método alternativo para la descripción de los datos. Existen, no obstante, diferencias, entre los medios de descripción de datos de ambos lenguajes; las más significativas se exponen en los puntos siguientes:

1. El COBOL describe los datos en la Data Division; el PL/I utiliza la sentencia DECLARE. La Data Division del COBOL contiene secciones separadas para las diferentes clases de datos; el PL/I no necesita separar en las sentencias DECLARE los diversos tipos de datos.
2. EJ COBOL necesita que todas las palabras proporcionadas por el programador se definan en la Data Division. El PL/l permite utilizar en los programas las palabras que proporcione el programador, sin estar definidas en una sentencia DECLARE; eJ significado de tales palabras se determina según el contexto, utilizándose un juego completo de reglas "por omisión" para determinar las características no especificadas de los datos.
3. En COBOL, la descripción de los datos en medios externos de almacenamiento se especifica en la Data Division. En PL/I, son las sentencias de entrada/salida las que especifican la descripción de los datos almacenados externamente (véase en el Capítulo 3 "Entrada/ Salida").
4. Las series de bits y os datos considerados como símbolos son tipos de datos en PL/l que no existen en COBOL.
5. **El** COBOL utiliza constantes figurativas; el PL/I, no.
6. El PL/I no impone ningún límite en el número de dimensiones de un conjunto dimensionado, ni en el de niveles de una estructura. El COBOL limita a un máximo de tres las dimensiones de una tabla (equivalente en COBOL al conjunto dimensionado del PL/I) (y a un máximo de 49 los niveles de un grupo (equivalente en COBOL a la estructura de PL/I). El orden de los nombres calificadores en COBOL es el inverso al utilizado en PL/I. El COBOL emplea las palabras clave lN y OF como operadores de calificación; el PL/I utiliza el punto.

3

Entrada / Salida

TABLA DE CONTENIDOS

INTRODUCCIÓN

Antes de que un ordenador pueda procesar los datos que están registrados en medios externos de almacenamiento, dichos datos deben reproducirse dentro del ordenador; este proceso de reproducción se llama entrada. Igualmente, al completarse el proceso, los datos procesados se hacen disponibles mediante su reproducción en un medio externo; este proceso de reproducción se llama salida: El presente capitulo estudia dos tipos principales de entrada/salida.

ATRIBUTOS DE ALMACENAMIENTO EXTERNOS

El estudio del capitulo anterior ha tratado sobre la descripción de los ítems de datos que se almacenan dentro del ordenador. Sin embargo, la mayoría de las aplicaciones de ordenadores se ocupan también de la representación de los datos en medios externos de almacenamiento. A través de dichos medios los datos se presentan al ordenador y se reciben del mismo. Cuando los datos se registran en medios externos se organizan en grupos llamados archivos. Por ejemplo, una colección de fichas de anotaciones de tiempo puede constituir un archivo en una aplicación de nómina. En cintas magnéticas, un archivo puede constar de los datos que en su tiempo se utilizarán para producir un informe impreso.

El ordenador transmite los datos a y desde un archivo mediante sentencias de entrada/salida. Sin embargo, si se concentra esencialmente la atención en los datos que se están transmitiendo, las características descriptivas del archivo tienen escasa importancia. El PL/I permite atribuir un nombre a un archivo y describir sus características con palabras claves llamadas atributos del archivo.

El estudio que sigue versa sobre los atributos de archivo de que dispone el PL/I.

ATRIBUTO FILE

El atributo FILE indica que el identificador asociado es un nombre de archivo. En la siguiente sentencia:

```
DECLARE MAESTRO FILE;
```

se declara que el identificador MAESTRO es un nombre de archivo.

ATRIBUTOS STREAM Y RECORD

Los atributos STREAM y RECORD describen la manera en que deben tratarse los datos del archivo. Indican el tipo de transmisión de datos (orientada a corriente o a registro) que pueden utilizarse en operaciones de entrada/salida para archivo (véase "Transmisión de Datos").

El atributo STREAM describe un archivo considerado como una corriente continua de ítems de datos en forma de caracteres.

El atributo RECORD describe un archivo que contiene un número de registros separados físicamente, constando cada registro de uno o más Ítems de datos en cualquier formato.

ATRIBUTOS INPUT, OUTPUT Y UPDATE

Para indicar el tipo de transmisión de datos que se permite para un archivo, puede especificarse uno de estos atributos (INPUT, OUTPUT o UPDATE). El atributo INPUT se utiliza para archivos que únicamente van a leerse. El atributo OUTPUT se utiliza para archivos que van a crearse; los archivos OUTPUT solo pueden escribirse. El atributo

UPDATE se utiliza para archivos ya existentes que van a leerse, o a los que van a añadirse nuevos registros, o en los que tienen que alterarse o suprimirse registros ya existentes.

En la siguiente sentencia:

```
DECLARE DETALLE FILE INPUT,
INFORME FILE OUTPUT,
MAESTRO FILE UPDATE;
```

DETALLE es el nombre de un archivo de entrada; INFORME es el nombre de un archivo de salida; MAESTRO es el nombre de un archivo que se emplea tanto para entrada como para salida.

ATRIBUTO PRINT

El atributo PRINT indica que se imprimirán los datos del archivo, es decir, que aparecerán como salida en una página impresa. El atributo PRINT solo puede declararse para archivos de salida. Un archivo con el atributo RECORD no puede contener el atributo PRINT.

ATRIBUTOS BUFFERED Y UNBUFFERED .

Los atributos BUFFERED y UNBUFFERED solamente se aplican a archivos que tienen el atributo RECORD. El atributo BUFFERED indica que los datos del registro que se está transmitiendo a o desde un archivo van a situarse en un buffer (es decir, en un área intermedia de almacenamiento). El atributo UNBUFFERED indica que los datos van a transmitirse directamente a o desde la posición especificada por un nombre de datos.

ATRIBUTO ENVIRONMENT

El atributo ENVIRONMENT se utiliza para especificar las características físicas de un archivo que depende del ordenador para el cual se escribe el programa en PL/I. El atributo se escribe de la siguiente forma:

```
ENNVIRONMENT (lista-de-opciones );
```

La lista de opciones se definirá individualmente para cada compilador PL/I. En la lista de opciones puede especificarse infamación tal como medios de archivo, formato de registros físicos, almacenamiento intermedio y disposición del archivo.

APERTURA Y CIERRE DE ARCHIVOS

Antes de procesarse un archivo mediante sentencias de transmisión de datos, deben tener lugar ciertas actividades de preparación del archivo, tales como comprobación de la disponibilidad de los medios externos de almacenamiento, posicionamiento de estos y asignación de los soportes apropiados de programación. Estas operaciones se conocen como apertura del archivo. Análogamente, al completarse el proceso, el archivo debe cerrarse. Cerrar un archivo consiste en dejar libres los recursos que se establecieron durante la apertura del archivo. El P/I dispone de dos sentencias OPEN y CLOSE para realizar estas funciones. Estas sentencias, sin embargo son optativas. Si para un archivo determinado no se ejecuta una sentencia OPEN, el archivo se abre automáticamente al ejecutarse la primera sentencia de transmisión de datos para este archivo; en tal caso, la preparación automática del archivo consiste en procedimientos Standard del sistema, utilizando la información que una sentencia DECLARE contiene sobre el archivo. Análogamente, si no se ha cerrado un archivo antes de completarse el programa el archivo se cierra automáticamente a la terminación del programa.

SENTENCIA OPEN

La sentencia OPEN obtiene y prepara archivos para las subsiguientes operaciones de entrada/salida.

La sentencia OPEN tiene el formato siguiente:

```
OPEN FILE (nombredearchivo) opciones;
```

Las opciones pueden tener lugar en cualquier orden y colocarse antes y después de la especificación FILE (nombredearehivo). Las opciones incluyen cualesquiera de los atributos de almacenamiento externo (excepto los atributos FILE y ENVIRONMENT) que se estudiaron en el Capitulo 2; la opción IDENT, que regula la lectura y escritura de registros de etiquetas de cabecera; las opciones PAGESIZE y LINESIZE, que controlan la disposición global de cada página de un archivo PRINT. Se estudian a continuación las opciones IDENT, PAGESIZE y LINESIZE.

OPCIÓN IDENT (INFORMACIÓN DE ETIQUETAS)

En una sentencia OPEN para un archivo de salida, la opción IDENT especifica que debe colocarse en el archivo un registro de etiqueta de cabecera. Para un archivo de entrada, IDENT proporciona jnformaci6n para la lectura de un registro de etiqueta de cabecera.

Para archivos de entrada, la información de etiquetas es el nombre de una serie de caracteres en que va a introducirse por lectura la etiqueta de cabecera. Para archivos de salida, la información de etiquetas es el nombre de una serie de caracteres que contiene el registro de etiqueta de cabecera que debe escribirse o bien es una constante de series de caracteres que debe escribirse como registro de etiqueta de cabecera. Para archivos de actualización, la opci6n IDENT especifica lectura de etiquetas pero no su escritura.

Para cada aplicación de PL/I, el compilador define el formato de los registros de etiqueta. Si no se especifica la opción IDENT, no se realiza ninguna operación de etiquetas. Considérese la siguiente sentencia:

```
OPEN INPUT FILE (TABLAS) IDENT (SERIE);
```

Esta sentencia abre el archivo de entrada llamado TABLAS, sitúa el archivo en su principio lógico e introduce por lectura el registro de etiqueta de cabecera en la sede de caracteres identificada por SERIES.

OPCIÓN PAGESIZE (W)

La opción PAGESIZE especifica la longitud de una página impresa. "w" indica el número de líneas contenidas en una página, incluyendo las líneas que se salten. Por ejemplo, la especificación PAGESIZE (45) significa que en una página no deben imprimirse o saltarse más de 45 líneas. Si se intenta empezar una nueva línea más allá de la 45, se empieza una nueva página, salvo que el programador haya especificado otra acción en una sentencia ON (la sentencia ON se estudia en el Capítulo 4; véase también la condición ENDPAGE en el Capitulo 4, "Condiciones ON").

La opción PAGESIZE solo puede especificarse para archivos que tengan el atributo PRINT.

OPCIÓN LINESIZE (W)

La opción LINESIZE especifica la longitud máxima de las líneas de una página iimpresa; "w" indica el número de posiciones de caracteres disponibles en una línea. Por ejemplo, la especificación LINESIZE(120) significa que la línea se extiende desde la posición de carácter 1 a la posición de carácter 120. Si se intenta colocar datos a la derecha de la posición de carácter 120, los datos en cuestión se colocan en la línea siguiente, empezando en la posición de carácter 1.

La opción LINESIZE solo puede especificarse para archivos que tengan el atributo PRINT.

Si se encuentra una sentencia OPEN para un archivo que ya ha sido abierto la sentencia se ignora.

SENTENCIA CLOSE

La sentencia CLOSE deja libres los recursos que se establecieron durante la apertura del archivo, vuelve a posicionar el archivo en su principio lógico, y origina la disposición correcta del archivo. Una sentencia CLOSE tiene la siguiente forma:

```
CLOSE FILE (nombredearehivo) IDENT (in-formación de etiquetas);
```

El significado de la opción IDENT es el mismo que en la sentencia OPEN, excepto que trata con registros de etiqueta de cola en lugar de registros de etiqueta de cabecera.

SI se encuentra una sentencia CLOSE y el archivo no ha sido abierto o ya ha sido cerrado, la sentencia se ignora.

TRANSMISIÓN DE DATOS

La función básica de la entrada y la salida es la trasmisión de datos, introduciendo los ítems de datos en el ordenador para su proceso y colocando los resultados de dicho proceso en medios externos de almacenamiento.

En PL/I se dispone de dos tipos de transmisión de datos:

- transmisión de datos orientada a registro y
- transmisión de datos orientada a corriente.

En la transmisión dle datos orientada a registro, los datos contenidos en el medio externo se consideran como una colección de registros físicamente separados, constando cada registro de uno o más ítems de datos en cualquier formato. Los datos se transmiten, un registro cada vez, de la misma forma que están registrados, bien interna o externamente; no se realiza ninguna conversión.

En la transmisión de datos orientada a corriente, los datos contenidos en el medio externo se consideran como una corriente continua de ítems de datos en forma de caracteres. Cada ítem de datos se convierte, si es necesario, a y de su formato interno apropiado, formato que se determina por los atributos del nombre de datos aL que está asignado el ítem de datos.

TRANSMISIÓN ORIENTADA A REGISTRO

La transmisión de datos orientada a registro trata con archivos que están compuestos por una serie de registros físicamente separados. Cada registro se lee o escribe como una unidad aislada, a o desde un almacenamiento intermedio direccionable o la posición especificada por un nombre de datos (normalmente el nombre de una estructura o de un conjunto dimensionado).

En la transmisión orientada a registro las principales sentencias que se utilizan para transmitir datos a o desde archivos de entrada y de salida son la sentencias READ y WRITE.

De los archivos a los que se hace referencia en las sentencias READ y WRITE debe declararse que tienen el atributo RECORD lo que especifica que el archivo debe utilizarse con sentencias orientadas a registro. El atributo UNBUFFERED especifica que los datos de los registros no ingresan en un almacenamiento intermedio direccionable, sino que se asignan directamente a o desde el nombre de datos especificado en la sentencia READ o WRITE.

Para un archivo sin almacenamiento intermedio, la sentencia READ tiene el siguiente formato:

```
READ FILE (nombrcdearchivo) INTO (nombre de datos);
```

La sentencia WRITE tiene el siguiente formato:

I

```
WRITE FILE (nombrcdearchivo) FROM (nombre de datos);
```

Al ejecutarse una sentencia READ, motivando que se lea un registro, no hay conversión de los tipos de datos para adaptarse a los atributos declarados para los nombres.

Los datos del registro deben ajustarse exactamente a la declaración del nombre al cual están asignados. Análogamente, un registro que se escribe tiene la misma forma externamente que su representación interna.

Cuando los registros van a introducirse por lectura en almacenamientos intermedios o escribirse en salida de los mismos en lugar de asignarse directamente a nombres de dato. la sentencia READ tiene una forma diferente:

```
READ FILE (nombredearchivo)   SET (variable de indicación);
```

La variable de indicación es un nombre que se utiliza para señalar la posición de almacenamiento intermedio. Está asociada con un nombre de datos por medio de una sentencia DECLARE. Dicho nombre de datos, se llama variable de base. Por ejemplo, considérese la sentencia siguiente:

```
DECLARE 1 DETALLE CONTROLLED (N),
          2 CTA_# CHARACTER (7),
          2 PAGO DECIMAL FIXED (6,2),
          2 NOMBRE CHARACTER (40);
```

En esta sentencia, DETALLE es una variable de base y N, una variable de indicación asociada con DETALLE. Como variable de base, DETALLE debe declararse como que tiene el atributo CONTROLLED (véase en el Capítulo 4 "Asignación de Memoria'" un estudio del atributo CONTROLLED). La sentencia DECLARE expresa que N indicará un almacenamiento intermedio en el que se introducirán los registros por lectura o desde el que se escribirán los mismos en salida, y que cada registro constará de tres Ítems de datos con los mismos atributos que los declarados para CTA_#, PAGO y NOMBRE. La ejecución de la sentencia READ que viene a continuación origina la introducción por lectura de un registro en este almacenamiento intermedio:

```
READ FILE (MAESTRO) SET (N);
```

La variable de indicación N señala ahora al principio del registro (en efecto, el valor de N es la dirección de la primera posición de memoria del almacenamiento intermedio). Es como si el registro se asignara directamente a DETALLE; una referencia a CTA_# viene a ser una referencia al primer ítem de datos del almacenamiento intermedio, una referencia a PAGO se convierte en una referencia al segundo ítem de datos de dicho almacenamiento y una referencia a NOMBRE en una referencia al tercer ítem de datos del almacenamiento intermedio.

Al ejecutarse la siguiente sentencia WRITE:

```
WRITE FILE (SALIDA) PROM (DETALLE);
```

se escribe un registro desde este almacenamiento (en el que se introdujo la sentencia por lectura).

También puede escribirse un registro desde un almacenamiento intermedio creado mediante una sentencia LOCATE. Esta sentencia tiene la forma:

```
LOCATE varjable de base FILE (nombredearchivo)
  SET (variable de indicación);
```

La sentencia LOCATE no motiva inmediatamente la escritura de datos. Expresa que va a asignarse un almacenamiento intermedio que tiene los mismos atributos que la variable de base especificada. La variable de indicación especificada señalará al almacenamiento intermedio. Subsiguientemente los datos se sitúan en el almacenamiento intermedio para formar un registro. El registro se escribe en el archivo de salida especificado inmediatamente antes de ejecutarse a siguiente sentencia LOCATE o WRITE situada delante del mismo archivo, o inmediatamente antes de formarse el archivo especificado.

Por ejemplo, supóngase que se ha leído un registro que contiene un número de cuenta, cantidad a pagar y nombre de un cliente, del modo descrito anteriormente. En el archivo de salida debe escribirse un registro que solamente contiene el nombre de cliente y la cal1tidad a pagar. Para establecer una variable de base y adjudicar un nuevo almacenamiento intermedio se necesitarían las siguientes sentencias:

```
DECLARE 1 REGISTRO_CUENTA CONTROLLED (p),
          2 CLIENTE CHARACTER (40),
          2 CANTIDAD DECIMAL FIXED (6,2);
 LOCATE REGISTRO_CUENTA FILE (SALIDA) SET (p);
```

El registro de salida se crea al mover los correspondientes ítems de datos del almacenamiento intermedio de entrada (denominados NOMBRE y PAGO) al almacenamiento intermedio de salida (los Ítems se denominan aquí CLIENTE Y CANTIDAD). El movimiento se realiza mediante el empleo de las sentencias de asignación que siguen (se muestran aquí las sentencias exactas a efectos del ejemplo; las sentencias de asignación se estudian a fondo en El Capitulo 5):

```
CLIENTE = NOMBRE;
CANTIDAD = PAGO;
```

Una vez creado, el registro permanece en el almacenamiento intermedio de salida hasta que la próxima sentencia LOCATE o WRITE para el archivo OUTPUT esté a punto de ejecutarse o hasta que el archivo SALIDA esté a punto de cerrarse.

La transmisión orientada a registro permite el empleo de archivos de actualización. Los datos se escriben en un archivo de actualización por medio de la sentencia REWRITE.

Cada registro que se lee se escribe de nuevo en el archivo de actualización, con o sin cambio antes de que se lea otro registro. Si el archivo de actualización no tiene almacenamiento intermedio, es decir, si los registros del archivo se asignan directamente a nombres de datos, la sentencia REWRITE tiene el formato siguiente:

```
REWRITE FILE (nombrcdcarchivo) PROM (nombre de datos);
```

Si el archivo de actualización tiene almacenamiento intermedio, es decir, si los registros se introducen por lectura en almacenamientos intermedios y se escriben en salida desde los mismos, solo es necesaria la siguiente sentencia para originar la escritura de un registro:

```
REWRITE FILE (nombrcdearchivo);
```

No es necesario especificar ningún nombre de datos porque la sentencia REWRITE siempre hace referencia al almacenamiento intermedio en el que se introdujo el registro por lectura.

TRANSMISIÓN ORIENTADA A CORRIENTE

La transmisión de datos orientada a corriente trata con archivos que se consideran como una corriente continua de datos en forma de caracteres. Los ítems de datos se transfieren desde la comente a los nombres de datos o desde estos a la corriente.

La transmisión orientada a corriente implica conversión de los datos. Todos los ítems de datos de la corriente están en forma de caracteres. En la entrada, se convierten automáticamente para adaptarse a los atributos del nombre de datos que se asignan; en la salida, los ítems de datos se convierten, si es necesario, a caracteres. Naturalmente, solo pueden realizarse aquellas conversiones de datos que son válidas (véase Capítulo 5 "Conversión entre Tipos de Datos").

Existen tres modalidades de transmisión orientada a corriente:

- transmisión controlada por edición,
- transmisi6n controlada por lista y
- transmisi6n controlada por datos.

Las tres modalidades utilizan las mismas sentencias para entrada y salida, las sentencias GET y PUT, respectivamente. Cualquiera que sea la modalidad utilizada, se requiere para cada sentencia GET o PUT la siguiente información:

- El nombre del archivo desde el que van a obtenerse los datos o al que estos van a transmitirse.
- Una lista de nombres de datos representativos de las áreas de memoria a las que van a transmitirse ítems de datos durante una operación de entrada, o desde las que se van a extraer los ítems de datos durante una operación de salida. Dicha lista se conoce como lista de datos.
- El formato de cada ítem de datos.

En ciertas circunstancias, toda esta información que se requiere puede estar sobreentendida; en otros casos, solo se necesita declarar explícitamente una parte de dicha información. Si no se especifica el nombre del archivo, se supondrán archivos Standard del sistema; esto se aplica a cualquiera de las tres modalidades de transmisión a corriente. En la transmisión controlada por lista y en la controlada por datos, no se especifica el formato de los ítems de datos.

En la entrada controlada por datos, no es necesario siquiera especificar la lista de datos.

TRANSMISIÓN DE DATOS CONTROLADA POR EDICIÓN

La transmisión controlada por edición especifica una descripción explícita de los ítems de datos tal como existen o están a punto de existir en la corriente de datos. Esta descripción explícita está contenida en una sentencia GET o PUT. La sentencia GET se utiliza para transmitir datos desde un almacenamiento externo a otro interno y la sentencia PUT para transmitir datos desde un almacenamiento interno a otro externo. Las sentencias GET y PUT utilizadas para transmisi6n controlada por edición tienen las formas siguientes:

```
GET FILE (nombre de archivo) EDIT (lista de datos) (lista de formatos);
```

```
PUT FILE (nombre de archivo) EDIT (lista de datos) (lista de formatos);
```

En sentencias GET o PUT controladas por edición, la lista de datos es una lista de nombres de datos a los cuales o desde los cuales van a transmitirse los ítems de datos que se encuentran en almacenamientos externos. Los nombres de datos en la lista están separados por comas.

La lista de formatos es una lista de descripciones de formatos separadas por comas. Hay dos tipos de descripciones de formatos:

- descripciones de formatos de datos y
- descripciones de formatos de control.

Las descripciones de formatos de datos describen los ítems de datos contenidos en la corriente de datos; las descripciones de formatos de control especifican operaciones de página, línea y espaciado.

Al ejecutarse una sentencia GET o PUT controlada por edición, los nombres de la lista de datos se emparejan con las descripciones de formatos de datos de la lista de formatos. Los datos se transmiten de la siguiente forma:

1. Si la sentencia es una GET, el ítem de datos descrito por la descripción del formato se adjudica al área identificada por el nombre de datos con el que está emparejada la descripción dél formato. Por ejemplo en la sentencia:

```
GET FILE (ARCHIVO_A) EDIT (PRIMERO)(A(20));
```

el nombre de datos PRIMERO se empareja con la descripción del formato A(20). El ítem de datos descrito por A(20) se adjudica al área del almacenamiento interno identificada por PRIMERO.

2. Si la sentencia es una PUT, el ítem de datos asignado al área identificada por el nombre de datos de la lista se sitúa en la corriente de datos. La representación de este valor en la corriente de datos depende de la descripción de formato con la que está emparejado el nombre de datos. Por ejemplo, en la sentencia:

```
PUT FILE (ARCHIVO_A) EDII (PRIMERO) (A(20));
```

el nombre de datos PRIMERO se empareja con la descripción del formato A(20). El ítem de datos identificado por PRIMERO se sitúa en la corriente de datos. La descripción del formato A(20) indica que el ítem de datos se representa en la corriente como una serie de 70 caracteres de longitud.

Obsérvese que no existe necesariamente ninguna correlación entre la forma de estar representado el mismo valor en memoria interna en la corriente de datos. La representación de un valor en memoria interna depende de los atributos del nombre de datos que aloja el valor; la representación de un valor en la corriente de datos depende de la descripción de su formato.

DESCRIPCIONES DE FORMATO DE DATOS.

Las descripciones de formato de datos se utilizan para describir la representación y características de los ítems de datos en la corriente de los mismos.

DESCRIPCIONES DE LOS DATOS DE SERIES DE CARACTERES

A(w)

La descripción de formato A(w) indica que el valor del ítem de la lista de datos con el que está asociada A(w) está representado en la corriente de datos como una serie de caracteres. La especificación indica el número de caracteres de la serie.

Considérese un ítem de datos llamado UNIDAD declarado con el atributo CHARACTER(20), significando que su representación interna es una secuencia de 20 caracteres. El valor de UNIDAD puede escribirse en un archivo llamado ARCHIVO_UNIDAD como una serie que consta de 20 caracteres mediante la siguiente sentencia:

```
PUT FILE (ARCHIVO_UNIDAD) EDIT (UNIDAD) (A(20));
```

En esta sentencia, la letra A indica que los datos deben escribirse como una serie de caracteres, y, el entero 20 especifica la longitud en caracteres de la serie. Si se hubiera utilizado la descripción de formato A(25), se hubiera añadido 5 caracteres en blanco al valor de UNIDAD, haciendo que la longitud del ítem escrito fuera de 25 caracteres. Analógicamente, de utilizarse la descripción de formato A(15), solo se escribirían 15 caracteres. Estos 15 caracteres contendrán el valor de los 15 primeros caracteres de UNIDAD.

Para operaciones de salida, la longitud de la serie de caracteres no necesita especificarse con el formato A. Si se omite, la longitud se determina según los atributos declarados del nombre con el que está emparejada la descripción de formato. Así, la sentencia PUT anterior podría haberse escrito como sigue:

```
PUT FILE (ARCHIVO_UNIDAD) EDIT (UNIDAD)(A);
```

Habiéndose declarado UNIDAD con el atribulo BIT (20) el valor de UNIDAD se convertiría de una serie de bits de 20 posiciones a una serie de caracteres de 20 posiciones compuesta por los caracteres numéricos cero y uno: y se imprimirían 20 caracteres. Conversiones similares se aplican a los ítems de datos almacenados internamente de punto fijo y de punto flotante. Si se hubiera declarado UNIDAD con el atributo FIXED (10), el valor de UNIDAD se convertiría de formato interno de punto fijo a formato externo de serie de caracteres, extendiéndose a la derecha con blancos, si es necesario, antes de imprimirse como una serie de caracteres de 20 posiciones.

La sentencia:

```
GET FILE (ARCHIVO_ÚNIDAD) EDIT (UNDAD) (A(20));
```

origina que los 20 siguientes caracteres del archivo llamado ARCHIVO_UNIDAD se adjudiquen a UNIDAD. El valor se transforma automáticamente de su representación n de caracteres especificada por el formato A(20) a cualquiera de las representaciones especificadas mediante los atributos declarados para UNIDAD.

• F(w,d)

La descripción de formato de punto fijo F(w,d) indica que el valor del ítem de la lista de datos con el que está asociada F(w,d) está representado en la corriente de datos como una serie de caracteres con un valor decimal que contiene un punto decimal supuesto o real.

La especificación w indica el número de posiciones de carácter de la serie. La especificación d indica que aparece un punto decimal supuesto o real de caracteres antes del final de la serie.

Cuando se escribe un ítem de datos desde el interior del ordenador, su valor se ajusta a la derecha del campo especificado por w en la descripción del formato. Si d es mayor que cero, aparecerá en el campo un punto decimal real antes de los d últimos caracteres. Si, en el ítem que se está escribiendo, el número le caracteres significativos vos a la derecha del punto decimal es menor que d, se rellenarán ceros a la derecha.

Cuando el valor del ítem que se está escribiendo es menor que cero, a la representación externa en caracteres del valor le antecederá un signo menos.

Si el valor es mayor o igual que cero no aparece ningún signo, por lo que la primera posición a la izquierda será un blanco. La especificación w debe incluir una posición para el signo menos y otra para el punto decimal.

Si el valor del ítem que se está escribiendo contiene ceros no significativos, estos se cambiarán por blancos; si el valor del ítem no rellena el campo especificado por la descripción de formato, aparecerán blancos a la izquierda.

AL introducirse por lectura un ítem de datos en él ordenador, el valor transmitido puede aparecer en cualquier sitio del campo especificado por w. Se supone la presencia en el campo de un punto decimal antes de los d últimos caracteres. No es necesario escribir la especificación d si el valor a leerse no va a contener posiciones decimales. Sin embargo, si aparece un punto decimal real en el valor leído y su posición contradice la especificada por d se ignorara dicha especificación d utilizándose la posición del punto decimal real.

Considérese el ítem de datos llamado CANTIDAD que tiene el atributo FIXED (4,2).Dichoq ítem puede escribirse en un archivo de salida Standard mediante la siguiente sentencia: '.

```
PUT EDIT (CANTIDAD) (F(6,2));
```

(6,2) especifica que debe aparecer un punto decimal antes de los dos últimos caracteres numéricos y que el número esté ajustado a la derecha en un campo de 6 caracteres. Los ceros a la izquierda se cambian por blancos y, si es necesario, se coloca un signo menos a la izquierda del primer carácter numérico.

Si, tal como se ha descrito anteriormente, CANTIDAD se utilizara en la sentencia:

```
GET EDIT (CANTIDAD) (F(6,2));
```

se explorarían los 6 caracteres siguientes de un archivo de entrada Standard en busca de un número decimal de punto fijo con dos posiciones decimales. Una vez localizado, se convertiría el número a una forma compatible con los atributos declarados para CANTIDAD.

Si CANTIDAD tuviera los atributos FIXED BINARY (24,7) que indican un número binario de punto fijo de 24 bits con siete posiciones binarias, la sentencia:

```
PUT EDIT (CANTIDAD) (F(6,2));
```

produciría en un archivo Standard de salida 6 caracteres conteniendo el valor de CANTI DAD convertido de su representación interna binaria de punto fijo a un número decimal representado como una serie de caracteres decimales con un punto decimal dos caracteres antes del final del ítem.

Una sentencia GET invertiría la conversión. Conversiones similares se aplicarían si los atributos de CANTIDAD fueran decimal o binario en punto flotante. Si CANTIDAD tuviera el atributo CHARACTER o una especificación de modelo de serie de caracteres, la conversión se aplicaría también, con tal que, en caso de salida, la serie de caracteres representara un valor numérico.

Si se declarara para CANTIDAD el atributo BIT, los datos de series de bits se tratarían como un entero binario, que se convertiría a un entero decimal, representándose como una serie de caracteres decimales.

• E(w,d)

La descripción de formato de punto flotante E(w,d) indica que el valor del ítem de la lista de datos con el que está asociado E(w,d) está representado en la corriente de datos como una serie de caracteres. Esta serie tiene un valor decimal y contiene un punto decimal supuesto o real y un exponente con signo precedido del carácter E.

La interpretación de w y d es la misma que en la descripción de formato F(w,d),con las siguientes excepciones:

- La especificación w debe incluir el número de caracteres ocupados por

- El exponente y su signo.
- Un signo para el valor, si éste es negativo.
- Un punto decimal real, si existe.

Cuando se utiliza esta descripción de formato, d indica el número total de caracteres entre el punto decimal y la E. En salida, aparecerá un punto decimal real a la izquierda de los dígitos significativos. En la sentencia:

```
GET EDIT (EVALUACION_BRUTO) (E(8,2));
```

el valor del ítem de la corriente de datos se adjudica a EVALUACION_BRUTO y se convierte automáticamente al formato interno especificado por los atributos EVALUACION_BRUTO.

DESCRIPCIONES DE FORMATOS DE SERIES DE BITS

• B(w)

La descripción de formato de serie de bits B(w) indica que el valor del ítem de la lista de datos con el que está asociado B(w) está representado en la corriente de datos como una serie de caracteres que contienen solamente los caracteres 1 y 0. La especificación w indica el número de caracteres de la serie.

Considérese un Ítem de datos llamado CODIGO que tiene el atributo BIT(40). Utilizando la siguiente sentencia:

```
PUT FILE (ARCHIVO_CODIGO) EDIT (CODIGO) (B( 40));
```

puede escribirse el valor de CODIGO, en un archivo llamado ARCHIVO_CODIGO, como una serie de 40 caracteres que representan bits.

Si en la secuencia anterior se hubiera utilizado la descripción de formato B(120), se añadirían 80 caracteres cero a la derecha de la serie.

En salida, no necesita especificarse la longitud de la serie con el formato B. De omitirse, la longitud se determina por los atributos declarados del nombre con el que está emparejada la descripción de formato.

Así la sentencia PUT anterior podría haberse escrito como sigue:

```
PUT FILE (ARCHIVO_CODIGO) EDIT (CODIGO) (B);
```

DESCRIPCIÓN DE FORMATO DE MODELOS

La descripción de formato de modelos tiene la siguiente forma:

```
P 'especificación del modelo'
```

La especificación de modelos se describe en el Capítulo 2, estudiándose en el 5 la compaginación mediante especificaciones de modelo.

La descripción de formato P puede utilizarse para describir valores de datos que están representados en la corriente de datos en formato de caracteres con o sin compaginación.

```
GET EDIT (TITULO, CUENTA) (P'AAAAA', P'9999');
```

Al ejecutarse esta sentencia, se transmiten los nueve caracteres siguientes de un archivo Standard de entrada y los cinco primeros caracteres (alfabéticos) se adjudican a TITULO; el entero situado en las cuatro posiciones de carácter siguientes (numéricas) se atribuyen

CUENTA. La representación interna de los datos se adaptará a los atributos, de TITULO y CUENTA.

Repetición de Descripciones de Formatos

Con Objeto de abreviar, puede aparecer un entero decimal inmediatamente delante de una descripción de formato para indicar e] número de veces que va a utilizarse dicha descripción antes de seleccionar la siguiente descripción de formato. La sentencia:

```
GET EDIT (UNIDAD, ITEM, COSTE) (2A(10), F(4,2));
```

tiene el mismo efecto que la sentencia:

```
GET EDIT (UNIDAD, ITEM, COSTE) (A(10), A(10), F( 4,2));
```

Para especificar el número de veces que va a utilizarse una descripción de formato, también puede utilizarse una expresión encerrada entre paréntesis, inmediatamente antes de la descripción de formato. Cuando la expresión tiene un valor cero o negativo, no se utiliza la descripción de formato asociada con ella.

Puede aplicarse un factor de repetición a más de una descripción de formato encerrado entre paréntesis las descripciones de formato y anteponiendo al paréntesis izquierdo el factor de repetición.

Así la sentencia:

```
PUT EDIT (UNIDAD, COSTE, ITEM, TASA)(2(A(10), F( 4,2)));
```

produce el mismo resultado que la sentencia:

```
PUT EDIT (UNIDAD, COSTE, ITEM, TASA) (A(10, F(4,2), A(10), F(4,2));'
```

Especificaciones Múltiples de Datos

En una sentencia GET o PUT, se llama especificación de datos controlada por edición a una lista de datos y a la lista de formatos asociada con ellos. Hasta ahora, los ejemplos de sentencias GET y PUT contenían cada uno una especificación de datos. En una sentencia GET o PUT puede escribirse más de una especificación de datos, cosa que suele hacerse a fin de simplificar la escritura de sentencias que emplean largas listas de datos de formatos. La relación entre un Ítem de formato y un nombre de datos se hace más evidente cuando las listas de datos y de formatos son cortas. Las sucesivas especificaciones de datos se separan por comas.

Por ejemplo, la sentencia

```
PUT EDIT (UNIDAD,COSTE.ITEM,TASA)
(A(10), F(4,2), A(10), F{4,2));
```

puede leerse más fácilmente como sigue:

```
PUT EDIT (UNIDAD, COSTE) (A(10), F(4,2))
(ITEM, TASA) (A(10), F{4,2));
```

Especificaciones de Datos para Estructuras V Conjuntos Dimensionados

Los nombres de una lista de datos pueden ser nombres de estructuras y de conjuntos dimensionados. Un nombre de estructura o de conjunto dimensionado en una lista de

datos sirve como representación concisa de todos los ítems elementales de la estructura o conjunto dimensionado.

Considérese la estructura siguiente:

```
DECLARE
   1 CUENTA
   2 ITEM    CHARACTER (10),
   2 NUMERO  FIXED (5),
   2 COSTE   FIXED (5,2);
```

La utilización de CUENTA en la sentencia:

```
PUT EDIT (CUENTA) (A(10), F(5),F(5,2));
```

equivale a escribir la siguiente sentencia:

```
PUT EDIT (ITEM, NUMERO, COSTE) (A(10),F(5),F(5,2));
```

Cuando en el ejemplo anterior se escribe la estructura CUENTA se requieren descripciones de formatos para cada ítem elemental de CUENTA, porque las sentencias GET y PUT transmiten los sucesivos ítems elementales.

En salida las listas de datos pueden especificar expresiones incluso expresiones de estructuras y conjuntos dimensionados. (Las expresiones se estudian en el Capítulo 5). Cada expresión, al evaluarse, se escribe en el formato especificado por un ítem de formato que está asociado.

Por ejemplo, la sentencia:

```
PUT EDIT ('TOTAL ES '|| 2*SUMA) (A(15));
```

convertirá el valor de 2*SUMA en una serie de caracteres, añadirá esta serie a la derecha de la constante 'TOTAL ES' y escribirá el resultado como una serie de caracteres de 15 posiciones en un archivo Standard de salida.

DESCRIPCIONES DE FORMATO DE CONTROL

Las descripciones de formato estudiadas hasta ahora se utilizan para describir ítems de datos tal como existen en la corriente de datos.

Las descripciones de formato que se estudian más adelante no son descripciones de ítems de datos; describen operaciones de espaciado y de impresión. Estas descripciones de formato de control se escriben en la lista de formatos de una sentencia GET o PUT junto con las descripciones de formato de datos estudiados previamente. Sin embargo, no están asociadas a nombres de datos en la lista de los mismos, de las sentencias GET o PUT.

DESCRIPCIÓN DE FORMATO DE ESPACIADO

Para indicar una operación de espaciado en una lista de formatos, se utiliza la siguiente descripción.

• X(w)

En entrada, el ítem de formato X especifica que los siguientes w caracteres de los datos en la corriente, van a pasarse por alto sin transmitirse.

En salida, el formato X especifica que deben insertarse en la corriente w caracteres en blanco. Por ejemplo, la sentencia:

```
GET EDIT (CUENTA, DEDUCCION) (A(5),X(5) A(5));
```

especifica que los 15 caracteres siguientes de un archivo Standard de entrada deben tratarse como sigue: los primeros cinco caracteres se adjudican a CUENTA, los cinco siguientes se pasan por alto ignorándose y los cinco caracteres restantes se asignan a DEDUCCION .

La sentencia:

```
PUT EDIT (ITEM, TOTAL) (A(4),X(2),F(5));
```

sitúa en un archivo Standard de salida cuatro caracteres que contienen el valor de ITEM, seguidos de dos blancos, seguidos de cinco caracteres que contienen el valor de TOTAL.

DESCRIPCIONES DE FORMATO DE IMPRESIÓN

Las descripciones de formato que siguen controlan operadores de impresión. Solamente se utilizan con archivos que tienen el atributo PRINT y, consecuentemente solo aparecen en sentencias PUT.

PAGE

La descripción del formato PAGE especifica Que el próximo ítem de datos que se escriba debe aparecer en una nueva página. Por ejemplo:

```
PUT EDIT (' CONTINUA EN LA PAGINA SIGUIENTE ')
(A(22)) ('INFORME MENSUAL DE VENTAS '
(PAGE, A(20));
```

Esta sentencia PUT da lugar a que la frase CONTINUA EN LA PAGIINA SIGUIENTE se escriba en la página actual y la cabecera INFORME MENSUAL DE VENTAS, en una nueva página.

• SKIP (w)

La descripción de formato SKIP (w) especifica que deben saltarse una o más líneas antes de escribirse el próximo ítem de datos. w indica la posición de la próxima línea respecto a la línea actual. La cantidad de líneas que se saltan es igual a w-1.

Por ejemplo la descripción de formato SKIP (5) significa que deben saltarse 4 líneas y que el próximo ítem de datos debe escribirse en la quinta línea a partir de la línea actual.

• LINE (w)

La descripción de formato LINE(w) especifica que el próximo ítem de datos debe escribirse en una línea determinada. w indica el número de la línea en la página. Por ejemplo la descripción de formato LINE(20) significa que el próximo ítem de datos debe escribirse en la línea 20 de la página.

• COLUMN (w)

La descripción de formato COLUMN(w) especifica posicionamiento horizontal, indicando la posición de carácter en que debe empezar el próximo ítem de datos. w es el número de posición para el primer carácter del ítem de datos.

Por ejemplo, la descripción de formato COLUMN(35) significa que el primer carácter del próximo ítem de datos debe aparecer en la posición de carácter 35 de la línea. Entre las descripciones de formato de impresión, la descripción de formato SKIP especifica posicionamiento relativo, mientras que LINE y COLUMN especifican posicionamiento absoluto.

El ejemplo que sigue muestra la utilización de descripciones de formato de impresión combinando unas con otras.

```
PUT EDIT ('INFORME BANCARIO EMPRESTITOS MENSUALES') (PAGE, LINE(2),A(24));
PUT EDIT (EMPRESTITO_ #, PRINCIPAL, INTERES, PAGO, SALDO) (SKIP(3), A(7),
COLUMN(15), F(8,2), COLUMN(35),F(3,3) COLUMN(45), F(6,2),COLUMN(65), F(8,2));
```

La primera sentencia PUT especifica que la cabecera INFORME BANCARIO EMPRESTITOS MENSUALES debe escribirse en la línea dos de una nueva página.

La segunda sentencia especifica que:

- deben saltarse dos líneas y que debe escribirse el valor de EMPRESTITO_ #, empezando en el primer carácter de la línea quinta;
- el valor de PRINCIPAL empezando en la posición de carácter 15;
- el de INTERES en la 35;
- el de PAGO en la 45;
- y el valor de SALDO en la posición de carácter 65. .

TRANSMISIÓN DE DATOS CONTROLADA POR LISTA

La transmisión de datos controlada por lista permite al programador especificar las variables a las que van a asignarse los datos (o desde las que van a tomarse) sin declarar específicamente ningún formato para los datos. El formato es Standard y lo suministra el compilador. La transmisión controlada por lista proporciona operaciones fáciles de entrada/salida para programadores que no necesitan formatos especiales en entrada o en salida y que están interesados solamente en una lista de los resultados del proceso.

La forma elemental de las sentencias GET y PUT, cuando se utilizan para entrada y salida controladas por lista es:

```
GET FILE (nombredearchivo) LIST (lista de datos);
PUT FILE (nombredearchivo) LlST (lista de datos);
```

FILE (nombredearchivo) es la especificación del archivo; LIST (lista de datos) es la especificación de los datos. El nombre del archivo y la lista de datos deben encerrarse cada uno entre paréntesis.

Las dos especificaciones no necesitan aparecer en un orden determinado.

Si se omite la especificación de archivo, se supone que debe utilizarse uno de los archivos Standard. En la transmisión controlada por lista, la palabra clave LIST debe siempre encabezar la especificación de los datos.

LISTAS DE DATOS CONTROLADAS POR LISTA

La lista de datos de una sentencia GET controlada por lista es una lista de variables (que representan áreas internas de memoria) a las cuales deben asignarse los ítems de datos de la corriente de datos. Las variables (nombres de datos) de una lista se separan por comas. El siguiente es un ejemplo de sentencia GET controlada por lista:

```
GET FILE (MAESTRO) LIST (PRESTAMO_ #, PRINCIPAL, TASA);
```

La sentencia GET del ejemplo anterior origina que se asignen a las variables de la lista de datos tres Ítems de datos del archivo MAESTRO en la secuencia en que están listadas; es decir, el primer ítem de datos se asigna a PRESTAMO_#, el segundo a PRINCIPAL y el tercero a TASA. La asignación cesa en este punto por haberse agotado la lista de datos.

La lista de datos de una sentencia PUT controlada por lista solamente difiere de la de una sentencia GET en que los ítems de datos pueden representarse mediante una expresión

distinta de su nombre; por ejemplo una expresión aritmética cuyo valor es el ítem que debe escribirse.

Una vez evaluado el valor representado por una expresión se transmite de la misma forma que el valor representado por una variable. Los ítems de la lista de datos incluyendo expresiones, si hay alguna, se separan por comas. El siguiente es un ejemplo de sentencia PUT controlada por lista:

```
PUT FILE (SALIDA) LIST (NOMBRE,6.3* TASA, NUMERO-10);
```

La sentencia PUT del ejemplo anterior origina que se escriban tres Ítems de datos en el archivo denominado SALIDA. La secuencia según la cual se escriben los ítems de datos obedece a la secuencia de dichos ítems en la lista de datos; es decir el primer ítem de dotas es el valor representado por la variable NOMBRE, el segundo es el valor que resulta de evaluar la expresión 6.3 *TASA, y el tercero es el valor resultante de evaluar la expresión NU MERO-10. La escritura cesa en este punto.

Obsérvese que, en la entrada o salida controlada por lista o en cualquier forma de transmisión orientada a corriente, es la lista de datos la que determina el número de ítems de datos extraídos de la corriente o insertados en la misma.

En entrada controlada por lista, los sucesivos ítems de datos del medio externo deben estar separados por comas o blancos. En salida, los blancos entre ítems se insertan, automáticamente.

REPRESENTACIÓN DE DATOS CONTROLADA POR LISTA

La representación interna y externa de un ítem de datos en una transmisión controlada por lista se determina por los atributos declarados para ella por el programador.

Por ejemplo, un ítem de datos para el cual se ha declarado el atributo CHARACTER (10) se registraría internamente como una serie de 10 caracteres de longitud. En salida, se escribirla de la misma forma. Para entender mejor cómo se aplica lo anterior a sentencias GET y PUT controladas por lista, supóngase que un archivo de entrada Standard contiene los siguientes datos:.

```
'NEW YORK', 'FEBRERO', -6.5, 72.6
```

Supóngase,. además, que aparecen en el programa las dos sentencias siguientes:

```
DECLARE  CIUDAD CHARACTER (12), MES CHARACTER (9),
         MINTEM FIXED DECIMAL (4,2),
         MAXTEM FIXED DECIMAL (5,2);
GET LIST (CIUDAD, MES, MINTEM ,MAXTEM);
```

La sentencia GET originaría la asignación de los ítems de datos de la forma siguiente:

- A CIUDAD se le asigna la serie de caracteres NEWYORK, ajustados a la izquierda y rellenados por la derecha con cuatro blancos.
- A MES se le asigna la serie de caracteres FEBRERO, ajustados a la izquierda y rellenados por la derecha con dos blancos.
- 3. A MINTEM se le adjudica el valor -06.50.
- 4. MAXTEM adopta el valor 072.60 .

Las series de caracteres se rellenan con blancos a la derecha para adaptarse a la longitud declarada de las series; las comillas no se conservan internamente. Los números decimales de punto fijo se alinean con el punto decimal supuesto, para adaptarse a la precisión declarada.

Considérese el resultado de la siguiente sentencia PUT:

```
PUT LIST (CIUDAD, MES, MAXTEM, MINTEM, 'RANGO:', MAXTEM - MINTEM);
```

El registro se imprimiría como:

```
NEW YORK FEBRERO 72.6-6.5 RANGO: 79.1
```

Nótese que al imprimirse una serie de caracteres, no se escriben las comillas sencillas, tanto si la serie se especifica como el valor de una variable (CIUDAD y M ES) como si se especifica como una constante de caracteres ('RANGO:'). Si se escribe una serie de caracteres en un archivo que no tiene el atributo PRINT, las comillas se insertan, si es necesario, y se escriben .

Cuando se escribe una serie de bits, aparecen las comillas sencillas, así como la letra B, incluso en archivos con PRINT. Los dígitos binarios se convierten a los caracteres 1 y 0.

TRANSMISIÓN DE DATOS CONTROLADA POR DATOS

Los formatos elementales de las sentencias GET y PUT, en transmisiones controladas por datos se escriben como sigue:

```
GET FILE (nombredearchiyo) DATA; ,
PUT FILE (nombredearchivo) DATA (lista de datos);
```

No es necesario incluir la lista de datos en la sentencia GET debido a que los ítems de datos en la corriente vienen acompañados de los nombres de datos a los que van a asignarse. Los datos en la corriente de entrada pueden tener una apariencia como esta:

```
A = 7.3B = 'ABCDE' c(4,2) = 9876;
```

El nombre de datos, seguido de un símbolo de asignación y del valor que va a asignarse, se llama una asignación. Las asignaciones en la corriente de entrada pueden separarse por comas o blancos. La última asignación que se obtiene de una sola sentencia GET debe ir seguida de un punto y coma. Si la corriente indicada anteriormente fuera parte de un archivo de entrada Standard del sistema, la sentencia:

```
GET DATA;
```

originaría que se asignaran valores a A, B y C.

En salida debe aparecer la lista de datos para especificar qué Ítem de datos deben escribirse en la corriente. La sentencia PUT, refiriéndose a los ítems de datos anteriores, podría ser:

```
PUT FILE (SALIDA) DATA (A, B, C (4,2));
```

Las asignaciones se separan por blancos en la corriente de salida, escribiéndose un punto y coma después del último ítem especificado en la lista de datos.

OPCION STRING

Una de las características de la transmisión de datos orientada a corriente se ocupa más de la transmisión interna de datos que de la entrada y la salida. En sentencias GET o PUT la opción FILE (nombre_de_archivo) puede sustituirse por la opción

```
STRING (nombre de serie).
```

Cuando se especifica, la opción STRING, la sentencia no tiene ninguna relación con un archivo. En una sentencia GET, la opción STRING indica que la serie designada debe

considerarse como una corriente de caracteres de entrada; en una sentencia PUT, indica que la serie designada debe considerarse como una comente de salida.

Aunque se puede utilizar la opción de serie con cualquiera de las tres modalidades de transmisión orientada a corriente, resulta más práctico utilizarla en asociación con una lista de formatos, puesto que los ítems individuales de la serie no necesitan separarse por comas o blancos.

Considérese el ejemplo siguiente:

```
GET STRING (REGISTRO) EDIT (NOMBRE, NUMERO_PAGO, HORAS, TASA)
                           (A(l2),A(7), F(2), F(4,2));
```

Esta sentencia especifica que va a explorarse la serie de caracteres que tiene el nombre REGISTRO, que está registrada en un área de memoria. Los 12 primeros caracteres de la serie deben asignarse a NOMBRE; los siguientes 7 caracteres deben asignarse a NUMERO_PAGO; los 2 siguientes deben convertirse a representación decimal de punto fijo y asignarse a HORAS y los 4 últimos caracteres especificados deben convertirse a un número decimal de punto fijo (con dos dígitos después del punto decimal) y asignarse a TASA. De existir otros caracteres en la serie, deben ignorarse.

La sentencia PUT con una opción de serie produce un efecto contrario. Considérese la sentencia:

```
PUT STRING (REGISTRO) EDIT (NOMBRE, NUMERO_PAGO, HORAS * TASA)
                           (A(l2), A(7), P'$$99.99');
```

Esta sentencia especifica que el valor de NOMBRE debe asignarse a las 12 primeras posiciones de carácter de la serie denominada REGISTRO, y Que el valor de NUMERO_ PAGO debe asignarse a las 7 siguientes posiciones de carácter de REGISTRO. El valor de HORAS debe multiplicarse por el de TASA y el producto debe convertirse a una serie de caracteres asignándose a las 7 siguientes posiciones de carácter de REGISTRO.

SENTENCIA DISPLAY

La sentencia DISPLAY produce la visualización de un mensaje al operador de la máquina. La unidad en la que se visualiza el mensaje se especificará para cada aplicación de PL/l.

El formato de la sentencia DISPLAY es:

```
DISPLAY (expresion) REPLY (nombre de datos);
```

La ejecuci6n de la sentencia DISPLAY produce la evaluación de la expresión y, cuando sea necesario, su conversión a una serie de caracteres. Esta serie de caracteres es el mensaje que va a visualizarse. En la opción REPLY el nombre de datos debe declararse como una serie de caracteres.

Este nombre de datos recibe el mensaje suministrado por el operador del ordenador. Se suspende la ejecución del programa hasta recibirse el mensaje del operador.

La especificación REPLY es optativa; cuando no se utiliza, la ejecución continúa sin interrupción. La siguiente sentencia:

```
DISPLAY ('¿CUAL ES EL PROXIMO ARCHIVO? ') REPLY (PROXIMO_ARCHIVO);
```

visualiza el mensaje: ¿CUAL ES EL PROXIMO ARCHIVO? y hace que el ordenador espere la respuesta del operador. La respuesta del operador se adjudica a PROXIMO_ARC,HIVO.

La sentencia:

```
DISPLAY ('FINAL DE LA FASE-2');
```

visualiza el mensaje indicado pero no interrumpe la ejecución del ordenador.

COMPARACIÓN ENTRE PL/I Y COBOL: ENTRADA / SALIDA

El estudio que sigue compara los medios de entrada/salida del PL/I y el COBOL. Ambos lenguajes emplean métodos similares para transmitir datos entre áreas internas y externas de almacenamiento hasta el punto de que las sentencias de entrada/salida procesan los archivos y los registros de identificación (registros de etiqueta) contenidos en los archivos pueden procesarse tanto en entrada como en salida. Los lenguajes difieren sin embargo, en la capacidad de entrada/salida; los puntos siguientes tratan de algunas de las diferencias más significativas.

- En COBOL la transmisión de datos implica archivos que están compuestos de registros lógicos. Uno o más ítems de datos forman un registro lógico; los datos se transmiten en la proporción de un registro lógico cada vez.
- El PL/I proporciona dos tipos de transmisión de datos.
  - La transmisión orientada a registro, como en el COBOL, trata con registros lógicos.
  - La transmisión orientada a corriente se ocupa de ítems de datos individuales; se considera a los archivos como corrientes continuas de ítems de datos, más que como conjuntos de registros lógicos.
- El PL/I proporciona especificaciones de formato de control que regulan las operaciones de impresión y espaciado.
- En PL/I, los registros de identificación (registros de etiquetas) se leen o escriben como resultado de la opción IDENT en una sentencia OPEN o CLOSE. En COBOL, los registros de etiquetas se especifican mediante una inscripción descriptiva del archivo en la Data Division, especificándose en una sentencia USE los procedimientos especiales para las etiquetas.
- El COBOL permite utilizar un nombre de archivo como calificador de nombres; el PL/I no lo permite.
- En PL/I) se especifican las características de un archivo en una sentencia DECLARE o OPEN. En COBOL, las características del archivo se especifican en una inscripción descriptiva del archivo en la Data Division.

4

Estructura del Programa

TABLA DE CONTENIDOS

INTRODUCCIÓN

Algunas de las sentencias de los programas en PL/I introducen por lectura datos, los escriben en salida realizan cálculos y efectúan las conversiones de una representación de datos a otra necesarias para hacer tales cálculos. Ciertas sentencias controlan el orden en que se ejecutan otras instrucciones del programa. Debido a que estas sentencias controlan el orden de ejecución de un programa, son las que definen su estructura. Estas sentencias especifican transferencias en el flujo de ejecución del programa, efectúan la preparación de ciertas partes del programa activando su ejecución, y especifican la ejecución iterativa de ciertas sentencias y el control del flujo del programa cuando ocurre un error.

El resto del presente capítulo estudia cómo se estructura un programa en PL/I, y explica la función de las sentencias que definen dicha estructura y controlan el flujo de ejecución del programa.

BLOQUES

Un programa en PL/I consta de un conjunto de procedimientos externos, cada uno de los cuales se compone de uno o más bloques. Un bloque es una recopilación o grupo de sentencias. Existen dos clases de bloques: bloques de procedimiento y bloques de comienzo.

BLOQUES DE PROCEDIMIENTO

Un bloque de procedimiento o, más resumido, un procedimiento, tiene el formato general:

```
símbolo: PROCEDURE.
  sentencia 1
    •
    •
    •
   sentencia n
END símbolo;
```

A la sentencia PROCEDURE debe preceder un símbolo. La sentencia END no necesita contener un símbolo ";" si lo tiene, el símbolo debe ser el mismo que el de la sentencia PROCEDURE.

BLOQUES DE COMIENZO

Un- bloque de comienzo tiene la forma general:

```
símbolo: BEGIN;
   sentencia 1
    •
    •
    •
   sentencia n
END símbolo;
```

Son dos las sentencias que, se requieren, una sentencia BEGIN y una END. Antes de la sentencia BEGIN puede escribirse un símbolo que la identifique como principio del bloque de comienzo. La sentencia END no necesita incluir un símbolo si lo incluye, el símbolo debe ser el mismo que el de la sentencia BEGIN.

BLOQUES INTERNOS Y EXTERNOS

Cualquier bloque de procedimiento o de comienzo puede incluir dentro de sí otro bloque completo de procedimiento o de comienzo. Un bloque debe estar completamente incluido en el otro. Los bloques pueden estar contenidos dentro de otros bloques. Cuando un bloque de procedimiento no está contenido en otro bloque se llama procedimiento externo. Un bloque de procedimiento incluido en otro bloque se llama procedimiento interno. No puede decirse de los bloques de comienzo que sean externos puesto que cada bloque de comienzo debe aparecer en algún otro bloque. Un programa consta de un conjunto de uno o más procedimientos externos que pueden contener bloques intemos (bloques jerarquizados).

El primer Bloque de procedimiento externo a ejecutarse en un programa debe identificarse mediante una palabra clave definida por la aplicación en el atributo OPTIONS. Esta palabra clave sigue a la palabra clave PROCEDURE en la sentencia PROCEDURE del bloque y se necesita incluso cuando el programa consta de un solo procedimiento externo.

Un ejemplo de programa que contiene bloques de procedimiento y bloques de comienzo es el siguiente:

```
A: PROCEDURE OPTIONS (MAIN);
   sentencia J
   B: BEGIN;
      sentencia 2
      sentencia 3
   END B;
   sentencia 4'
   C:  PROCEDURE;
       sentencia 5
       D: BEGIN;
          sentencia 6
          sentencia 7
       END D;
       sentencia 8.
    END C; .
    sentencia 9
END A;
```

En este ejemplo, el bloque A es un procedimiento externo que contiene el bloque de comienzo B y el procedimiento interno C. El bloque D es un bloque de comienzo contenido en el procedimiento C. La palabra clave definida por aplicación MAIN identifica al bloque A como el primer bloque a ejecutarse. Aunque los bloques de comienzo y los de procedimiento tienen una semejanza física, difieren en un aspecto funcional importante. Un bloque de comienzo como una sentencia simple, se ejecuta en el transcurso del flujo secuencial del programa. Con un procedimiento, sin embargo, el flujo del programa se salta el procedimiento, desde la sentencia anterior a la PROCEDURE hasta la sentencia que sigue a la END de dicho procedimiento. La única forma en que puede ejecutarse un procedimiento interno es mediante una sentencia CALL. La forma elemental de una sentencia CALL es:

```
CALL  nombre_de_inscripción;
```

El nombre de inscripción es el símbolo de una sentencia PROCEDURE y, en consecuencia, el nombre de un procedimiento. La sentencia CALL produce la transferencia del control del flujo del programa al procedimiento denominado mediante el nombre de inscripción Una sentencia CALL en un bloque de procedimiento o comienzo que origine la transferencia del control del flujo del programa a otro procedimiento se conoce como sentencia CALL "activante". El procedimiento al que se ha transferido el control se conoce como procedimiento "activador'. Al completarse la ejecución de un procedimiento activado, el control se envía a la sentencia que sigue a la sentencia CALL activante.

Puesto que un procedimiento se puede activarse utilizando u nombre como nombre de inscripción en una sentencia CALL activante, cada sentencia PROCEDURE debe tener un símbolo. En el ejemplo anterior, la sentencia 1 podría ser una sentencia CALL que activara el procedimiento C; se escribiría:

```
CALL e;
```

A la terminación del procedimiento C, el control se enviaría a la sentencia:

```
B: BEGIN;
```

por ser esta sentencia la que sigue a la sentencia CALL activante.

ÁMBITO DE LAS DECLARACIONES

El mismo nombre puede aparecer en varios bloques y describirse con atributos diferentes en cada bloque. Esto permite al programador dividir un programa en muchos bloques y utilizar el mismo nombre en cada uno para ítems de datos diferentes. Por ejemplo:

```
NOMINA: PROCEDURE;
        DECLARE TARIFA CHARACTER (5);
         •
         •
    INFORME: BEGIN;
        DECLARE TARIFA_FIXED (4,2);
          .
          .
          .
          .
    END INFORME;
END NOMINA;
```

NOMINA es un procedimiento externo e INFORME es un bloque de comienzo interno a NOMINA. El nombre de datos TARIFA se describe de forma diferente en cada uno de estos bloques. TARIFA con el atributo CHARACTER (5) se aplica a todo el procedimiento NOMINA, excepto para el bloque de comienzo INFORME. En INFORME el nombre de datos TARIFA tiene distinto significado y se aplica a un ítem de datos diferente.

La región de un programa dentro de la cual se aplica la descripción de un nombre se llama ámbito de la declaración o ámbito del nombre establecido por la declaración.

En el ejemplo anterior, el ámbito de la declaración:

```
DECLARE TARIFA CHARACTER (5);
```

se extiende a lo largo de todo el procedimiento NOMINA , pero sin incluir el bloque INFORME. El ámbito de la declaración:

```
DECLARE TARIFA FIXED (4,2);
```

se limita al bloque INFORME. En general, las declaraciones separadas del mismo nombre implican nombres exclusivos con ámbitos distintos que no se solapan.

Puede resultar necesario, sin embargo, utilizar el mismo Ítem de datos en bloques diferentes. El PL/I proporciona distintas formas de conseguir lo anterior, mediante la jerarquización de bloques, utilizando el atributo EXTERNAL y empleando nombres como parámetros y argumentos. Cada uno de estos métodos será estudiado a su vez.

BLOQUES JERARQUIZADOS

Si los bloques están jerarquizados puede utilizarse el mismo ítem de datos en bloques diferentes. Cuando se declara un nombre de datos en un bloque exterior mediante una sentencia DECLARE y no se declara de nuevo en un bloque interno el ámbito de la declaración incluye el bloque interno.

Considérese el ejemplo siguiente:

```
x: PROCEDURE;
    DECLARE NUMERO FIXED (3), COSTE FIXED (4,2);
      •
      •
      •
   GET LIST (NUMERO, COSTE);
      •
      •
      •
END x;
y:  PROCEDURE;
    DECLARE TITULO CHARACTER (10);
      •
      •
    PUT LIST (TITULO, NUMERO, COSTE);
      •
      •
      •
END y;
```

Al ejecutarse el procedimiento X, NUMERO Y COSTE , se establecen como decimales de punto fijo, asignándoles valores mediante la sentencia GET. El ámbito de NUMERO y COSTE incluye el procedimiento "Y". Las referencias a estos dos nombres de datos en la sentencia PUT emplean la declaración establecida en el procedimiento x. El ámbito de TITULO, sin embargo, se limita al procedimiento y; cuando el control deja al procedimiento y los datos de TITULO no son accesibles mediante sentencias del procedimiento x. Consecuentemente, el símbolo TITULO del procedimiento y no debe ser utilizado por sentencias da procedimiento x.

La transferencia del control desde el procedimiento x al procedimiento y debe hacerse, por medio de una sentencia CALL. De haberse escrito el procedimiento y como un bloque de comienzo, no sería necesaria ninguna sentencia CALL; se permitiría que el flujo secuencial del control prosiguiera en la sentencia BEGIN del bJoque Y.

ATRIBUTOS EXTERNAL E INTERNAL

Si no se hubieran jerarquizado los procedimientos x e y del ejemplo anterior y fueran a utilizarse en ambos procedimientos los mismos valores de NUMERO y COSTE, el medio de indicarlo sería declarar NUMERO y COSTE en ambos procedimientos con el atributo EXTERNAL. Esto se ilustra escribiendo de nuevo el ejemplo anterior como sigue:

```
x: PROCEDURE;
   DECLARE NUMERO FIXED (3) EXTERNAL, COSTE EXTERNAL FIXED (4,2);
    •
    •
    •
   GET LIST (NUMERO, COSTE);
    •
    •
    •
```

```
END x;

y: PROCEDURE;
   DECLARE TITULO CHARACTER (10),
           NUMERO EXTERNAL FIXED (3),
           COSTE FIXED (4,2) EXTERNAL;
         •
         •
         •
    PUT LIST (TITULO, NUMERO, COSTE)¡ •
      •
      •
END Y;
```

Este ejemplo tiene el mismo efecto respecto a NUMERO y COSTE que el ejemplo anterior empleando bloques jerarquizados.

De no haberse utilizado para NUMERO el atributo EXTERNAL en los dos procedimientos anteriores, el símbolo NUMERO en el procedimiento x, y el procedimiento y, se referiría a dos ítems de datos separados y distintos.

Cuándo se emplea el mismo nombre de datos en dos o más procedimientos externos para referirse al mismo ítem de datos, debe declararse el nombre de datos con el atributo EXTERNAL en cada procedimiento. En todas estas declaraciones del mismo nombre de datos, los atributos declarados deben ser compatibles, puesto que las declaraciones implican un solo Ítem de datos.

La utilizaci6n del atributo EXTERNAL no está restringida a bloques de procedimiento; también puede aparecer en bloques de comienzo que sean externos el uno con respecto al otro.

Cuando no se declara el atributo EXTERNAL para un nombre de datos se supone el atributo INTERNAL. Sin embargo se supone que los nombres de archivos y de símbolos de bloques externos de procedimiento tienen el atributo EXTERNAL (los nombres de símbolos de procedimientos externos se estudian más adelante en "Parámetros de Nombres de Inscripción y el Atributo ENTRY"). El atributo INTERNAL especifica que el ámbito de un nombre es el bloque para el cual la declaración del nombre es interna (en páginas anteriores de este capítulo aparece un estudio sobre el ámbito de las declaraciones). Los atributos EXTERNAL e INTERNAL se llaman atributos de ámbito y también se estudian en este capítulo, en "Asignaci6n de Memoria",

PARÁMETROS Y ARGUMENTOS

Otra forma de emplear los mismos ítems de datos en procedimientos diferentes consiste en especificar una lista de nombres entre paréntesis en una sentencia PROCEDURE siguiendo a la palabra PROCEDURE. Los nombres de esta lista se llaman parámetros.

. Los parámetros aparecen en un sentencia PROCEDURE de la siguiente forma:

```
símbolo: PROCEDURE (parámetro 1, ... parámetro n);
      sentencia 1
        •
        •
        •
      sentencia n
 END símbolo;
```

Los parámetros nunca están permitidos en sentencias BEGIN y pueden declararse en un procedimiento con una sentencia DECLARE, pudiendo ser nombres de datos, de archivos y de inscripciones (nombres identificativos de procedimientos). Los parámetros tienen que declararse en el procedimiento al cual se aplican, no es posible declararlos en cualquier otro bloque. Si no se da ninguna declaración explícita, se supone una declaración implícita o contextual, aplicable al procedimiento que contiene los parámetros.

Los parámetros permiten la programación, de procedimientos que requieren los valores de nombres de datos desconocidos. Considérese un procedimiento que es utilizado por otros varios procedimientos para analizar una deducción de impuestos. La persona que escribe el procedimiento posiblemente no conozca cómo llaman otros programadores en sus procedimientos al ítem de datos de deducción de impuestos. Sin embargo, la persona que escribe el procedimiento de deducción de impuestos puede proseguir asignando al ítem de deducci6n de impuestos el nombre de datos que prefiera. Puede, por ejemplo, utilizar el nombre de datos DEDUCCION_IMPUESTOS. Este nombre se coloca en la lista de parámetros del procedimiento, pudiendo utilizarse también en las sentencias del procedimiento. Otro programador puede llamar D_I al item de deducción de impuestos en su procedimiento. Aunque el ítem de deduoción de impuestos tiene dos nombres diferentes en dos procedimientos distintos, se puede establecer una asocioct6n entre D_I y DEDUCCION_IMPUESTOS. Esto se realiza colocando D_I en una lista de nombres entre paréntesis en la sentencia CALL que transfiera el control al procedimiento de deducci6n de impuestos. Los nombres de esta lista se llaman argumentos. Los argumentos aparecen en la sentencia CALL de la siguiente forma:

```
CALL nombre de inscripcion (argumento1, ....., argumenton);
```

Los argumentos pueden ser nombres de datos, nombres de archivo y nombres de inscripciones. Los parámetros de un procedimiento activado y los argumentos de la sentencia CALL activante se emparejan uno a uno de izquierda a derecha. El número de argumentos debe ser igual al de parámetros.

Durante la ejecución de un procedimiento activado, el nombre de cada parámetro se hace equivalente al nombre de su argumento asociado. En general, los atributos de un argumento deberían ser los mismos que los de su correspondiente parámetro; los argumentos que identifican conjuntos dimensionados deben corresponderse con los parámetros que identifican tales conjuntos, y los argumentos que identifican estructuras deben también corresponderse con los parámetros que las identifican.

Considérese el siguiente procedimiento:

```
SALIDA: PROCEDURE (TITULO, NUMERO, COSTE);
    DECLARE TITULO CHARACTER (10),
            NUMERO FIXED (3),
            COSTE FIXED (4,2);
      .
      .
      .
   PUT LIST (TITULO, NUMERO, COSTE);
     .
     .
     .
END SALIDA;
```

Este Procedimiento contiene tres parámetros: TITULO, NUMERO Y COSTE, cuyos valores se escriben en un archivo estándar como salida controlada por lista. Supóngase que el procedimiento SALIDA se activa desde el siguiente procedimiento:

```
PROCESO: PROCEDURE;
    DECLARE NOMBRE CHARACTER (10),
            CUENTA FIXED (3),
            PRECIO FIXED (4,2);
       .
       .
       .
     CALL SALIDA (NOMBRE, CUENTA, PRECIO);
       .
       .
       .
END PROCESO;
```

Al transferirse el control al procedimiento SALIDA por medio de la sentencia CALL del procedimiento PROCESO, los valores de los argumentos NOMBRE, CUENTA Y PRECIO son los valores de los parámetros correspondientes TITULO, NUMERO y COSTE. Una vez concluido el procedimiento SALIDA, el control vuelve a la sentencia que, sigue a la CALL del procedimiento PROCESO.

Cuando un argumento es el nombre de un conjunto dimensionado, el número de dimensiones y el de ítems de cada dimensión del conjunto deben corresponder a los del parámetro asociado.

Cuando un argumento es el nombre de una serie de caracteres o de bits, la longitud de la serie debe ser la misma que la del parámetro correspondiente. Sin embargo, es posible que en el momento de escribirse el procedimiento no se conozca el número de ítems de las dimensiones del conjunto ni la longitud de una serie. En este caso, puede utilizarse en la declaración del parámetro, un asterisco (*) para cada dimensión o para la longitud de la serie. El asterisco indica entonces que la longitud de la serie o el número de ítems de una dimensión es el mismo que para el argumento correspondiente.

Considérese el ejemplo siguiente:

```
IMPRESION: PROCEDURE (TABLA_TARIFA);
    DECLARE TABLA_TARIFA(*) FIXED (4,2);
       .
       .
    PUT LIST (TABLA_TARIFA);
       .
       .
       .
END IMPRESION;
```

Este procedimiento contiene un parámetro TABLA_ TARIFA que es el nombre de un conjunto unidimensional. El asterisco que aparece en la declaración indica que el tamaño

del conjunto no está especificado y que este asumirá el tamaño del argumento correspondiente en la sentencia CALL.

Supóngase que el procedimiento IMPRESION se activa desde el bloque siguiente:

```
BEGIN;
DECLARE TABLA_TRABAJO (100) FlXED (4,2);
  •
  •
  •
CALL IMPRESION (TABLA_TRABAJO);
  •
  •
  •
END;
```

Al transferirse el control al procedimiento IMPRESION por medio de la sentencia CALL del bloque de comienzo, la longitud de TABLA_TARIFA se convierte en la de TABLA_TRABAJO, que en este caso es 100, y los valores de TABLA_TRABAJO se convierten en los de TABLA_TARIFA. Cada vez que se activa el procedimiento IMPRESION, puede asociarse TABLA_TARIFA con un argumento de diferente tamaño. Sin embargo, el conjunto denominado mediante el argumento debe ser unidimensional y los elementos del conjunto deben tener el atributo FIXED (4,2).

Además de nombres de datos, de archivos y de inscripciones, los argumentos también pueden ser expresiones. Las expresiones se estudian en el Capítulo 5.

Por ejemplo, la sentencia que sigue:

```
CALL TARIFA (12 * SALARIO_MENSUAL);
```

podría utilizarse para ejecutar un procedimiento denominado TARIFA que empleara un parámetro representativo de un salario anual. El valor de la expresión:

```
12* SALARIO_MENSUAL
```

sería entonces el valor del parámetro empleado por el procedimiento TARIFA.

PARÁMETROS DE NOMBRES DE INSCRIPCIÓN Y EL ATRIBUTO ENTRY

Se ha definido un nombre de inscripción como una constante de símbolo que sigue directamente a la palabra CALL en una sentencia CALL. También pueden utilizarse los nombres de inscripción en listas de parámetros y de argumentos. Cuando en una sentencia CALL se utiliza como tal un nombre de inscripción, se considera que está declarado contextualmente como nombre de inscripción; cuando se utiliza en la lista de argumentos de una sentencia CALL, no se considera que está declarado contextualmente. La descripción contextual se estudia en el Capítulo 2 en Atributos por Omisión.

Por ejemplo, considérese el siguiente procedimiento:

```
RUTINAPROCEDURE (x, y, z);
   DECLARE x FIXED DECIMAL (4),
           Z LABEL;
     •
     •
     •
   CALL y;
     •
     •
     •
END RUTINA;
```

La sentencia DECLARE de este procedimiento especifica que el parámetro x es un entero decimal de punto fijo de cuatro dígitos y que el parámetro z es un nombre de símbolo que puede utilizarse en el cuerpo del procedimiento, y está declarado contextualmente como nombre de inscripción. Si se activa RUTINA mediante una sentencia CALL en el siguiente procedimiento:

```
PROCESO: PROCEDURE;

  CALL RUTINA (CUENTA, EDICION, L5);

END PROCESO;
```

Resulta entonces necesario que el argumento CUENTA tenga los atributos del parámetro x, que EDICION sea un nombre de inscripción y L5 un símbolo de sentencia. La aparición de EDICION como argumento en la sentencia CALL no lo convierte en un nombre de inscripción por contexto. Salvo que EDICION se haya utilizado en una sentencia CALL como nombre de inscripción, debe declararse explícitamente como tal en el procedimiento PROCESO. Esto se realiza mediante el atributo ENTRY como sigue:

```
PROCESO: PROCEDURE;
     DECLARE CUENTA FIXED DECIMAL (4);
    EDICION ENTRY,
    L5 LABEL;
      •
      •
      •
    CALL RUTINA (CUENTA, EDICION, L5);
      •
      •
      •
END PROCESO;
```

En el procedimiento PROCESO, se define el nombre RUTINA contextualmente como nombre de inscripción, porque sigue inmediatamente a la palabra clave CALL.

SECUENCIA DE CONTROL

Cuando se está ejecutando un programa, su secuencia de control determina el orden de ejecución de las sentencias. Dentro de un bloque, el control suele pasar secuencialmente de una sentencia a la siguiente. Si se encuentra una sentencia DECLARE, el control la salta, pasando a la próxima sentencia. La ejecuci6n secuencial de sentencias es modificada, sin embargo, por las siguientes sentencias:

- CALL, PROCEDURE, END, RETURN, GOTO, IF, DO y ON.

De ellas, las tres primeras ya se han mencionado. RETURN se estudiará en el párrafo siguiente. Seguirá un estudio sobre la activación y rescisi6n de bloques, para estudiar después las sentencias GOTO, IF, DO y ON. Puesto que la asignación de la memoria en núcleos viene influenciada por la secuencia de control de un programa, este capítulo incluye también, en su parte final, un estudio sobre asignación de memoria.

SENTENCIA RETURN

Las sentencias PROCEDURE y END delimitan bloques de procedimiento de forma que el control secuencial se salta los bloques de procedimientos internos hasta la sentencia que sigue a la END de dicho bloque de procedimiento interno. La sentencia CALL se utiliza para transferir el control a un procedimiento especificado. Al llegar el control a la sentencia END de un procedimiento, el control vuelve a la sentencia que sigue a la CALL. Es posible devolver el control antes de llegar a la sentencia END de un procedimiento. La sentencia RETURN sirve a este propósito y se escribe:

```
RETURN;
```

Considérese el siguiente procedimiento:

```
PRUEBA: PROCEDURE;
     •
     •
     •
  RETURN;
     •
     •
     •
  RETURN;
     •
     •
     •
END PRUEBA;
```

Cada vez que en este procedimiento se encuentren sentencias RETURN o la sentencia END, se devuelve el control a la sentencia que sigue a la CALL que activó el procedimiento PRUEBA.

ACTIVACIÓN Y RESCISIÓN DE BLOQUES

Se dice que un bloque de comienzo se activa cuando el control pasa por la sentencia BEGIN del bloque. Se dice que un bloque de procedimiento se activa cuando una sentencia CALL transfiere el control a la sentencia PROCEDURE del bloque. Existen diversas formas de rescindir un bloque activo:

1. Se rescinde un bloque de comienzo cuando el control pasa por la sentencia END del bloque.
2. Se rescinde un bloque de procedimiento al ejecutarse una sentencia RETURN o END del bloque.

3. Se rescinde un bloque a la ejecución de una sentencia GO TO que transfiere el control a un punto no contenido en el bloque. (La sentencia GO TO se estudia más adelante en este Capítulo.)

SUBORDINACIÓN DINÁMICA (INCLUSIÓN) DE BLOQUES

Al rescindirse un bloque, se rescinden todos los subordinados dinámicos de ese bloque. Considérense en el estudio que sigue los dos bloques: A y B.

Se dice que el bloque B es subordinado dinámico inmediato del bloque A si B se activa mediante una sentencia interior a A. Una sentencia es interior al bloque A si está contenida en A pero no en cualquier otro bloque que a su vez esté contenido en A.

Considérese el siguiente ejemplo:

```
 R: PROCEDURE;
     sentencia 1
     sentencia 2
  s: PROCEDÚRE;
     sentencia 3
     sentencia 4
    T: BEGIN;
       sentencia 5
END T;
  sentencia 6
END s;
    sentencia 7
END R;
```

En este ejemplo, las sentencias 1, 2 Y 7 son interiores a R por estar contenidas en R y no en otro bloque que a su vez esté contenido en R. Las sentencias 3, 4 Y 6 no son interiores a R pero lo son a S. La sentencia 5 solamente es interior al bloque T.

El bloque B puede ser subordinado dinámico inmediato del A de las siguientes formas:

- B es un bloque de procedimiento inmediatamente contenido en A, (es decir, D no está contenido en ningún otro bloque que a su vez lo esté en A) y se menciona mediante una sentencia CALL interior a A.
- B es un bloque de procedimiento no contenido en A que se menciona mediante una sentencia CALL interior a A.
- B es un bloque de comienzo que se ejecuta como consecuencia de una interrupción. (Las interrupciones se estudian más adelante en este capítulo con la sentencia ON).

A su vez, el bloque B puede tener un subordinado dinámico inmediato C, etc., de forma que se crea una cadena de bloques (A, B, C, D) en la que son activos todos los bloques. En esta cadena, los bloques C, D, ... son subordinados dinámicos de A, pero solamente B es subordinado dinámico inmediato de A.

Al rescindirse el bloque A, todos los bloques activos contenidos en B también se rescinden.

SENTENCIA GO TO

La sentencia GO TO transfiere el control a la sentencia que se especifique. Hay dos formatos de la sentencia GOTO:

```
GO TO constante considerada como símbolo;
CO TO nombre de símbolo;
```

En el primer formato, el control se envía a la sentencia que tiene como símbolo la constante. En el segundo formato la sentencia GO TO tiene el efecto de un conmutador de salidas múltiples; el control se transfiere a la sentencia identificada por el valor actual del nombre de símbolo. Puesto que el nombre del símbolo puede tener diferentes valores a cada ejecución de la sentencia GO TO, el control puede no transferirse siempre a la misma sentencia. El nombre de símbolo puede ser también el nombre de un conjunto dimensionado de símbolos. El ejemplo que sigue' muestra el empleo de una sentencia GO TO que efectivamente es un conmutador de salidas múltiples.

```
SWITCH: PROCEDURE;
          •
          •
          •
        DECLARE L LABEL (L1, L2) INITIAL (L2);
        GO TO ENCUENTRO;
        x = y - 1;
        L = L2;
        GO TO ENCUENTRO;
        Y =x – 1;
        L = Ll;
ENCUENTRO: CALL ERROR (x ,Y ,z);
           IF Z = LIMITE THEN GO TO L;
               •
               •
END SWITCH;
```

Cuando el valor de L es L2, la sentencia GO TO envía el control a la sentencia CALL. Cuando el valor de L es L1, el control se envía a la primera sentencia de asignaci6n.

El valor del nombre de símbolo se cambia a L1 01 ejecutarse la sentencia de asignación L = L1. (Las sentencias de asignaci6n se estudian en el Capítulo 5).

Una sentencia GO TO no puede transferir el control a un bloque que no haya sido activado o haya sido suspendido o rescindido.

Una sentencia GO TO que transfiere el control desde un bloque incluido en una jerarquía de bloques a otro de nivel superior en la jerarquía que comprende otros bloques rescinde todos los bloques que sean subordinados dinámicos del bloque al que se transfiere el control.

En el ejemplo siguiente:

```
PRIMERO: PROCEDURE;
          •
          •
          •
       SEGUNDO: BEGIN,
            •
            •
           TERCERO: BEGIN;
                 •
                 •
                 •
                 CALL CUARTO;
                    •
                    •
             CUARTO: PROCEDURE;
                    •
                    •
                    •
                 GO TO SEGUNDO:
                    •
                    •
                    •
             END CUARTO;
                 •
                 •
                 •
           END TERCERO;
             •
             •
             •
         END SEGUNDO;
             •
             •
             •
END PRIMERO;
```

La sentencia GO TO SEGUNDO envía el control al bloque de comienzo llamado SEGUNDO y rescinde los bloques CUARTO y TERCERO.

SENTENCIA IF

Con frecuencia es aconsejable ejecutar una sentencia o serie de sentencias solamente bajo ciertas condiciones. En tal situación, puede resultar conveniente evaluar una expresión y, sobre la base de esta evaluación, seleccionar o rechazar la sentencia o sentencias a ejecutarse. El PL/I proporciona para este fin la sentencia IF, junto con ocho operadores de comparación que se utilizan para formar expresiones de comparación que pueden ser verdaderas o falsas. Estas expresiones se emplean en la sentencia IF para determinar si deben ejecutarse o no una o varias sentencias.

EXPRESIONES DE COMPARACIÓN

Las expresiones de comparación utilizan los siguientes operadores de comparación:

```
>       (mayor que)
¬>      (no mayor gue)
>=      (mayor o igual que)
=       (igual que)
¬=      (no igual que)
<       (menor que)
¬<      (no menor que)
<=      (menor o igual que)
```

Estos operadores se utilizan con nombres de datos y constantes, al objeto de formar expresiones de comparar. Por ejemplo, la expresión:

```
CUENTA > 10
```

es una expresión de comparación en la que el valor numérico de CUENTA se compara con la constante 10. Si el valor de CUENTA es mayor que 10, la expresión es verdadera; de otra forma es falsa.

```
NOMBRE < 'RODERTO'
```

es verdadera cuando el valor de serie de caraccteres de NOMBRE sea menor que la constante de serie de caracteres 'ROBERTO', de otra forma es falsa. Las series de caracteres se comparan de izquierda a derecha, carácter a carácter. La comparación se basa en una secuencia de intercalación definida al objeto. Si las series de caracteres son de longitudes diferentes, la serie más corta se considera prolongada a la derecha con blancos.

En las expresiones de comparación están permitidas series de bits. La siguiente expresión:

```
MASCARA ¬= '1010101'B
```

es verdadera si el valor de serie de bits de MASCARA no es igual que la constante de serie de bits '1010101'B. La expresi6n implica una comparación de dígitos binarios de izquierda a derecha. El dígito binario cero es menor en comparación al dígito binario uno. Si las series de bits son de longitud diferentes, la más corta se considera prolongada a la derecha con bits cero. En el Capítulo 5 aparece un estudio más detallado de las expresiones de comparación.

El resultado de una comparación es una serie de un bit de longitud; el valor de una comparación verdadera es la constante '1'B; una comparación falsa tiene el valor '0'B.

EXPRESIONES DE COMPARACIÓN EN SENTENCIAS IF

Una sentencia IF puede adoptar uno de 2 formatos:

```
IF expresión de comparación THEN unidad
IF expresión de comparación THEN unidad 1
ELSE unidad 2
```

Cada "unidad" es una sentencia sencilla, un bloque de comienzo o un grupo. Un grupo consta de una secuencia de sentencias y/o bloques precedidos de una sentencia DO y rescindidos por una sentencia END.

Un ejemplo de grupo es la siguiente secuencia de sentencias:

```
DO; GET LIST (BRUTO, NETO);
CALL BENEFICIOS;
END;
```

Este grupo puede aparecer en una sentencia IF como sigue:

```
IF VENTAS = 22500 THEN DO; GET LIST (BRUTO, NETO);
CALL BENEFICIOS;
END;
```

En este ejemplo, si el valor numérico de VENTAS es igual a la constante 22500, se ejecuta el grupo de sentencias que sigue a la palabra clave THEN; en caso contrario se salta dicho grupo, pasando el control a la sentencia posterior a la END del grupo.

Cuando la unidad es una sentencia sencilla, no es necesario especificar las sentencias DO y END, salvo que se desee iteración por medio de la sentencia DO (estudiada más adelante). La siguiente sentencia es un ejemplo de "unidad" de sentencia sencilla.

```
IF BENEFICIOS < 0 THEN GO TO PERDIDAS;
```

Cuando el valor numérico de BENEFICIOS es menor que cero, se envía el control a la sentencia de símbolo PERDIDAS; de otra forma, se ejecuta la sentencia que sigue a GO TO PERDIDAS.

En el segundo formato de la sentencia IF una comparación verdadera origina la ejecución de la unidad posterior a THEN, motivando que se salte la unidad posterior a ELSE. Cuando la comparación es falsa, se salta la unidad posterior a THEN, ejecutándose la unidad posterior a ELSE. Considérese el ejemplo siguiente:

```
IF VENTAS > 1000
   THEN BEGIN;
          DECLARE BRUTO FIXED (7,2),
          NETO FIXED (6,2);
          GET FILE (DETALLE) EDlT
             (BRUTO, NETO) (F(7,2),F(6,2));
          CALL ANALISIS (BRUTO, NETO);
          PUT FILE (INFORME) LIST (BENEFICIOS);
        END;
ELSE DO;
         CALL IMPUESTOS (VENTAS);
         GO TO AJUSTE;
END;
```

Cuando el valor numérico de VENTAS es mayo que 1000, se ejecuta el bloque de comienzo posterior a THEN, saltándose el grupo DO posterior a ELSE; en caso contrario, se ejecuta el grupo DO y se salta el bloque de comienzo.

Las unidades que siguen a THEN y ELSE en una sentencia IF pueden también contener una o más sentencias y, en este caso, las unidades que siguen a THEN y ELSE se tratan

emparejadas. Cada unidad 2 ELSE se empareja con la unidad 1 THEN inmediatamente precedente que no esté emparejada. Por lo tanto resulta necesario especificar una unidad 2 ELSE para cada unidad 1 THEN. En este caso, la unidad 2 puede ser una sentencia ELSE "nula", es decir, venir seguida la palabra ELSE por un punto y coma.

El ejemplo que sigue ilustra el empleo de sentencias IF jerarquizadas, una de las cuajes emplea la cláusula ELSE nula.

```
IF EDAD> 21
     THEN
        IF PESO < 150
           THEN IF CABELLO = 'CASTAÑO'
                   THEN GO TO TIPO 1;
                ELSE GOJO TIP02;
        ELSE;
ELSE GO TO TIP03;
```

En este ejemplo, el efecto de la sentencia ELSE nula es el de no ejecutar nada y transferir el control a la sentencia que sigue a GO TO TIPO3 sí PESO < 150 es falsa y EDAD > 21, verdadera.

SENTENCIA SELECT

Proporciona un mecanismo de alternativa múltiple con una sola salida, para evitar la anidación de IF's. Contiene una sentencia SELECT, uno o varios WHEN para las expresiones que son evaluadas y un OTHERWISE, opcional, para el resto de casos sin evaluar.

El formato de la sentencia es:

```
SELECT (EXPRESION);
    WHEN (EXP1,EXP2,EXP3) ACCION1;
    WHEN (EXP4,EXP5)  ACCION2;
    OTHERWISE ACCION_N;
END;
```

En este caso se evalúa EXPRESION y el resultado se almacena, para compararlo con cada una de las expresiones de los WHEN, en el momento que una de ellas se cumple, se ejecuta la acción correspondiente al WHEN y se termina la sentencia.

```
SELECT;
    WHEN (EXP1) ACCION1;
    WHEN (EXP2)  ACCION2;
    OTHERWISE ACCION_N;
END;
```

En este caso a la primera expresión que sea verdadera se ejecuta la acción del when correspondiente.

Si el OTHERWISE es omitido y en la ejecución del SELECT, no hay ninguna condición que se satisfaga, la condición ERROR se propaga.

Todas sentencias se pueden ejecutar en acción excepto: DECLARE, END, ENTRY, FORMAT, PROCEDURE o %sentencias.

SENTENCIA DO

La sentencia DO, utilizada junto con una sentencia END, proporciona el medio de agrupar un conjunto de sentencias. Permite también la ejecución repetida de una secuencia de sentencias. Así como la modificación y prueba de ítems de datos para controlar la repetición. El hecho de poder repetir la ejecución de un conjunto de sentencias el número de veces que se especifique suele dar por resultado programas más pequeños y eficientes. Una sentencia DO puede especificarse de diversas formas.

Considérese el siguiente formato de una sentencia DO:

```
DO WHILE (expresión de comparación);
    sentencia 1
      •
      •
      •
    sentencia n
END;
DO UNTIL (expresión de comparación);
    sentencia 1
      •
      •
      •
    sentencia n
END;
```

Este uso de la sentencia DO motiva que se ejecute repetidamente la secuencia indicada de sentencias, en tanto que el valor de la expresión de comparación sea verdadero. (Las expresiones de comparación se han estudiado someramente con la sentencia IF . Cuando la expresión de comparación sea falsa el control se transfiere a la sentencia que sigue a END. En el siguiente ejemplo en que la sentencia DO se encuentra en primer lugar:

```
DO WHILE (CODIGO = 1);
     GET L1ST (CODIGO, SERIE);
     PUT EDIT (SERIE) (A);
END;
```

el valor numérico de CODIGO se compara con 1. Si la comparación es igual se ejecutan las sentencias GET y PUT, devolviéndose el control a la sentencia DO, en la que, de nuevo se compara con 1 el valor de CODIGO. Cuando la comparación no es igual, se transfiere el control a la sentencia que sigue a la END. Cada vez que se lee un registro los datos del primer campo del registro se asignan a CODIGO debiendo tener un valor numérico de 1. El último registro procesado deberá contener en el primer campo un valor numérico distinto de 1.

Otro formato de la sentencia DO es el siguiente:

```
DO nombre de datos = expresión_1 TO expresión_2 BY expresión 3;
         sentencia 1
            •
            •
            •
        sentencia n
END;
```

En este formato de la sentencia DO las expresiones son expresiones aritméticas que representan valores numéricos. (Las expresiones numéricas se estudian con detalle en el Capítulo 5). Al encontrarse la sentencia DO en primer lugar, el nombre de datos adopta inicialmente este valor de la expresión 1, valor que se compara con el de la expresión 2; si el resultado de la comparación cae dentro del rango de valores de la expresión 1 y de

la 2, se ejecuta la secuencia de sentencias. La sentencia END devuelve entonces el control a la sentencia DO, donde el valor del nombre de datos se incrementa en el valor de la expresión 3. Si el nuevo valor del nombre de datos cae fuera dél límite especificado en la expresión 2, el control se envía a la sentencia que sigue a la END; en caso contrario se ejecuta la secuencia de sentencias, transfiriendo de nuevo el control la sentencia END a la sentencia DO.

El ejemplo que sigue ilustra este formato de la sentencia

```
DO CONTADOR = 1 TO l0 BY 1;
    GET FILE (MAESTRO) EDIT (ITEM(CONTADOR))(A(80));
END;
```

En este ejemplo, el nombre de datos CONTADOR se utiliza como subíndice en la sentencia GET para especificar una posición del conjunto denominado ITEM. Al encontrarse en primer lugar la sentencia DO, el valor de CONTADOR se pone a 1, ejecutándose la sentencia GET. Los 80 caracteres siguientes del archivo MAESTRO se asignan a la primera posición del conjunto dimensionado ITEM; después, la sentencia END devuelve control a la sentencia DO. El valor de CONTADOR se incrementa en 1 y se prueba para determinar si excede del rango 1 a 10. La segunda ejecución de la sentencia GET asigna los 80 caracteres siguientes del archivo MAESTRO a la segunda posición del conjunto ITEM. Este proceso se repite hasta que el valor de CONTADOR excede a 10.

En este formato de la sentencia DO, las expresiones pueden tener valores negativos. Por-estar permitidos dichos valores negativos, no necesita ser mayor la expresión 2 que la 1. pudiendo utilizarse el valor de la expresión 3 como decremento más que como incremento.

Los dos formatos anteriores de la sentencia DO pueden combinarse como sigue:

```
DO nombre de datos = expresión 1 TO expresión 2 BY expresión 3
WHILE (expresión 4);
     sentencia 1
         •
         •
         •
     sentencia n
END;
```

Cuando se especifica de este modo, se ejecuta repetidamente la secuencia de sentencias hasta que tenga lugar una de las siguientes circunstancias:

- o el valor del nombre de datos cae fuera del rango especificado,
- o la expresión de comparaci6n asociada con WHILE pasa a ser falsa.

El ejemplo que sigue ilustra la forma combinada de la sentencia DO:

```
DO INDICE = 10 TO 1 BY -l
    WHILE (DEDUCCION < CALCULO);
         •
         •
END;
```

En este ejemplo el valor de INDICE oscila de 10 a 1 en decrementos de 1. Cuando

- el valor de INDICE cae fuera del rango 10 a -1 o
- la expresión de comparación DEDUCCION < CALCULO demuestra ser falsa,

se envía el control a la sentencia que sigue a la END.

Es posible transferir el control a un grupo DO desde el exterior de dicho grupo solamente si el grupo DO está delimitado por una sentencia DO que no especifica ejecución repetida.

SENTENCIA ON

Durante el transcurso de la ejecución de un programa, puede darse una cualquiera de cierto conjunto de condiciones que pueden provocar una interrupción.

Una interrupción motiva la suspensión de las actividades normales del programa, a fin de permitir la ejecución de una acción especial.

Cuando se ha realizado la acción especial, la ejecución del programa puede, o no, proseguir en el punto donde fue suspendida. En general, el lugar del programa donde puede ocurrir una interrupción es imprevisible.

Para la mayoría de las condiciones que pueden causar una interrupción, el programador puede especificar la acción especial a realizar. Para hacer esto, puede especificar la condición en una sentencia ON; por lo tanto, estas condiciones se llaman condiciones ON. Más adelante figura un estudio de las condiciones individuales ON. A cada condición-ON se le da un nombre exclusivo que sugiere la condición. Por ejemplo, ZERODIVIDE denomina la condición que resulta de intentar dividir por cero. Estos nombres (llamados en lo sucesivo condiciones-ON) son una parte intrínseca del lenguaje, aunque el programador puede utilizarlos para otros propósitos (tales como nombres de archivos, de datos o de etiquetas) en tanto que no exista ninguna ambigüedad.

Los formatos generales de una sentencia ON son:

```
ON condición SNAP unidad;
ON condición SYSTEM;
```

El primer formato permite al programador fijar qué acción va a realizarse cuando tenga lugar la condición especificada. La acción se especifica en la "unidad"; se escribe como una sentencia sencilla sin símbolo o como un bloque de comienzo sin símbolo. Obsérvese que no puede aparecer en la unidad una sentencia RETURN. La palabra clave SNAP es optativa; si se especifica, produce un listado de la información pertinente al estado del programa cuando tiene lugar la condición especificada.

El segundo formato especifica que va a realizarse una acción estándar del sistema. (Más adelante se estudia la acción estándar del sistema para cada condición). El ejemplo siguiente:

```
ON ZERODIVIDE CALL ANALISIS;
```

especifica que debe ejecutarse la sentencia:

```
CALL ANALISIS;
```

cuando se intente dividir por cero. Si al intentar dividir por cero debe realizarse una acción estándar del sistema, puede emplearse la siguiente sentencia:

```
ON ZERODIVIDE SYSTEM;
```

UTILIZACIÓN DE LA SENTENCIA ON

La sentencia ON es una sentencia ejecutable. Al ejecutarse una sentencia ON interior a un bloque (por ejemplo, el bloque B) establece una acción ON. Si, por concurrir la condición especificada en la sentencia ON, se interrumpe cualquiera de las sentencias ejecutadas después de la ejecución de la sentencia ON y antes de la rescisión del bloque B (incluyendo la ejecución de las sentencias de todos los subordinados dinámicos del bloque B), la sentencia o bloque de comienzo que aparece en la sentencia ON se ejecuta como si fuera activada como un bloque de procedimiento. El control se devolverá normalmente al punto que siga a la interrupción.

Si, después de establecida una acción determinada por la ejecución de una sentencia ON y mientras la especificación de esta acción está todavía en vigor, se ejecuta otra sentencia ON que especifique la misma condición, la última sentencia ON tomará vigencia según se ha descrito anteriormente de forma que su acción especificada determinará la acción de interrupción para la condición determinada. Si tanto una como otra de estas sentencias ON (antigua y nueva) son interiores al mismo bloque el efecto de la sentencia ON antigua queda invalidado completamente. Cuando la nueva sentencia ON está en un bloque subordinado dinámicamente del bloque que contiene la sentencia ON antigua, la efectividad de la sentencia ON antigua se suspende temporalmente. Esta efectividad se restaura a la cancelación o rescisi6n del bloque que contiene la nueva sentencia ON.

La acción estándar del sistema para condiciones ON se establece automáticamente. En algunas situaciones, al programador puede interesarle especificar su propia acción paro una condición dada aplicable a una parte de la ejecución del programa, e invalidar después esta especificación, permitiendo que tenga lugar la acción estándar del sistema. El ejemplo siguiente ilustra cómo puede realizarse lo anterior.

```
A: PROCEDURE;
       •
       •
       •
   ON ZERODIVIDE CALL ANALISIS;
       •
       •
       •
   ON ZERODIVIDE;
       •
       •
       •
   ON ZERODIVIDE SYSTEM;
       •
       •
       •
END A;
```

Al transferirse el control al procedimiento A, el sistema "permite" la condición ZERODIVIDE; cualquier condición ZERODIVIDE que tenga lugar antes de ejecutarse la primera sentencia ON ZERODIVIDE producirá una interrupción, con acción estándar del sistema. Sin embargo, la ejecución de la primera sentencia ON ZERODIIDE establece la acción especificada por la sentencia CALL. Cualquier interrupción ZERODIVIDE motivará que se ejecute esta acción hasta que se ejecute la segunda sentencia ON ZERODIVIDE. La acción especificada aquí es una sentencia nula; las subsiguientes interrupciones ZERODIVIDE serán ignoradas eficazmente hasta que el control llegue a la tercera sentencia ON ZERODIVIDE, que restablecerá la acción estándar del sistema.

PREFIJOS

Un prefijo es una lista de nombres de condición) separados por comas y encerrados entre paréntesis. Un prefijo puede agregarse a una sentencia. Al estar agregado a una sentencia, el prefijo precede a la sentencia completa, incluyendo cualquier posible símbolo de la sentencia, y va seguido del signo dos puntos (:) para separarlo del resto de la sentencia. Las sentencias con prefijo tienen el formato general:

```
(nombre de condición 1, …, nombre de condición n):
símbolo: sentencia.
```

En un prefijo pueden aparecer los siguientes nombres de condición:

```
UNDERFLOW
OVERFLOW
ZERODIVIDE
FIXEDOVERFLOW
CONVERSION
SIZE
SUBSCRIPTRANGE
```

Cada nombre de condici6n puede ir precedido de la palabra NO; por ejemplo, en la lista de prefijos puede especificarse NOOVERFLOW.

OBJETO DE LOS PREFIJOS

Las condiciones denominadas en el prefijo de una sentencia pueden tener lugar durante]a ejecución en el programa de una sentencia que caiga dentro del ámbito del prefijo (más adelante se estudia el ámbito de los prefijos).

Si realmente surge una de estas condiciones, la aparición en el prefijo del correspondiente nombre de condición -o su negación con la palabra NO- determina si tendrá lugar o no una acción de interrupción.

Una condición puede o puede no causar una interrupción, dependiendo de que la condición sea o no "permitida". El "permiso" de las condiciones denominadas UNDERFLOW, OVERFLOW, ZERODIVIDE, FIXEDOVERFLOW y CONVERSION la suministra automáticamente el PL/I; cualquier aparición de una de estas condiciones originará una interrupción, a no ser que se haya negado el "permiso" mediante el uso de un prefijo que contenga el nombre de la condición precedido de la palabra NO. Al programador le corresponde "permitir" las condiciones denominadas SIZE y SUBSCRIPTANGE mediante el uso de un prefijo. Por ejemplo, no tendrá lugar ninguna interrupción para un error SIZE, salvo que el error surja en una sentencia que esté dentro del ámbito de un prefijo SIZE.

ÁMBITO DE LOS PREFIJOS

Se conoce como ámbito de un prefijo la parte de un programa para la que se "permite" una condición mediante dicho prefijo. El ámbito del prefijo depende de la sentencia a la cual esté agregado.

Si la sentencia es una PROCEDURE o BEGIN, el ámbito del prefijo es el bloque definido por esta sentencia, incluyendo todos los bloques jerarquizados, excepto aquellos para los que se vuelve a especificar la condición por medio de otro prefijo. El ámbito no incluye los bloques de procedimientos externos que son subordinados dinámicos de los bloques interiores de este ámbito.

Cuando la sentencia sea un IF o un ON, el ámbito del prefijo no incluye las unidades (bloques o grupos) que forman parte de la sentencia. Cualquiera de tales unidades puede, a su vez, tener un prefijo agregado, las reglas de cuyo ámbito se ajustan a las que se establecen más adelante.

Para las sentencias que no sean IF, ON, PROCEDURE o BEGIN, el ámbito de los prefijos es el de la sentencia misma. El ámbito incluye cualquier expresión que aparezca en la sentencia, pero no incluye los procedimientos explícitamente activados por la sentencia. Considérese el siguiente ejemplo:

```
(SIZE): A: PROCEDURE;
             •
             •
             •
           ON SlZE GO TO ERROR_A;
             •
             •
             •
           CALL B;
         END A;
(SIZE, NOOVERFLOW): B: PROCEDURE;
          •
          •
          •
       ON SIZE GO TO ERROR_B;
          •
          •
          •
        RETURN;
      END B;
```

En este ejemplo, el prefijo (SIZE) "permite" la condición SIZE para el procedimiento A y especifica que si tiene lugar un error de tamaño durante cualquier cálculo en el procedimiento A, tendrá lugar una interrupción. El prefijo (SIZE, NOOVERFLOW) del procedimiento B especifica la misma necesidad con respecto a un error SIZE en el mismo procedimiento; además, especifica para el procedimiento B la supresión de cualquier interrupción que pudiera originarse por una condición de OVERFLOW.

Después de empezar la ejecución del procedimiento A y antes de ejecutarse la primera sentencia ON, cualquier error de tamaño se traducirá en una interrupción con acción Standard del sistema. Después de ejecutarse la primera sentencia ON y antes de]a ejecución de la sentencia ON del procedimiento B activado, cualquier error SIZE provocará una interrupción, con ejecución de la sentencia GO TO ERROR_A. Después de ejecutarse la sentencia ON del procedimiento B, la sentencia GO TO ERROR_ B queda establecida para la condición SIZE, aunque el efecto de la sentencia ON previa se suspende solo temporalmente. Después de ejecutarse la sentencia RETURN del procedimiento B, se restablece la efectividad de la sentencia ON previa, de forma que los errores SIZE que surjan después de este punto provocan de nuevo la ejecución de la sentencia GO TO ERROR_A.

Si durante la ejecución del procedimiento A tiene lugar cualquier condición de "overflow", se producirá una interrupción, con acción Standard del sistema para la condición OVERFLOW. Sin embargo, la interrupción quedará suprimida ante cualquier aparición de una condición "overflow" durante la ejecución del procedimiento B.

En el ejemplo que sigue:

```
(NOOVERFLOW): A: PROCEDURE;
                  •
                (OVERFLOW): B: BEGIN;
                              •
                              •
                              •
                            END B;
                              •
                              •
             END A;
```

se suprimirán las interrupciones para condiciones de "overflow" que ocurran durante la ejecución de la parte del procedimiento A que no está incluida en el bloque B. Las condiciones de overflow que tengan lugar durante la ejecución del bloque B provocarán una interrupción.

CONDICIONES ON

El estudio que sigue presenta todos los nombres de condición utilizados para identificar una condición-ON y explica las circunstancias bajo las que la condición tiene lugar, la acción Standard del sistema especificada por el PL/I que se llevarla a cabo en ausencia de una acción especificada por el programador y, cuando sea aplicable, el resultado de cualquier cálculo afectado por la condici6n.

CONVERSION

Esta condición tiene lugar cuando la conversión (en proceso interno o durante entrada/salida) de un tipo de datos a otro produce resultados erróneos; no debe hacerse hipótesis alguna sobre el resultado de la conversión; la acción Standard del sistema consiste en listar un comentario y provocar la condición ERROR.

ENDFILE (NOMBRE DE ARCHIVO)

Esta condición surge durante operaciones GET o READ al intentar leer más allá del delimitador del archivo especificado y la acción Standard del sistema consiste en listar un comentario y motivar la condición ERROR.

ENDPAGE (NOMBRE DE ARCHIVO)

Esta condición aparece durante operaciones PUT al intentar empezar una nueva línea después del límite especificado por la opción PAGESIZE de la sentencia OPEN y la condición solamente tiene lugar una vez por página de forma que pueden Imprimirse líneas adicionales en la página como resultado de una acción especificada por el programador; la acción Standard del sistema consiste en empezar una nueva página y continuar el proceso.

ERROR

Esta condición surge cuando una situación de error fuerza la suspensión del proceso; la acción Standard del sistema consiste en provocar la condici6n FINISH.

FINISH

Esta condición tiene lugar inmediatamente antes de que el proceso llegue a su terminación normal; la acción Standard del sistema consiste en terminar el proceso.

FIXEDOVERFLOW

Esta condición aparece cuando el resultado de una o creación aritmética de punto fijo excede el tamaño máximo definido por la aplicación; el resultado se trunca por la izquierda al tamaño máximo; la acción Standard del sistema consiste en listar un comentario y proseguir el proceso.

OVERFLOW

Esta condición surge cuando el exponente de un número en punto flotante excede al tamaño máximo definido por la aplicación; el resultado es el valor positivo máximo; la acción Standard del sistema consiste en listar un comentario y motivar la condici6n ERROR.

SIZE

Esta condición tiene lugar cuando hay pérdida de bits o dígitos de orden superior, motivada por la asignaci6n de un valor de punto fijo a un nombre de datos declarado con el atributo FIXED; la condición SIZE depende del tamaño declarado del nombre de datos y no del tamaño máximo para los números de punto fijo definido por la aplicación: no debe hacerse hipótesis alguna sobre el resultado de la asignación; la acción Standard del sistema consiste en listar un comentario y provocar la condición ERROR.

SUBSCRIPTRANGE

Esta condición tiene lugar al evaluar un subíndice y encontrar que cae fuera de sus límites especificados; no debe hacerse hipótesis alguna sobre el resultado de la evaluación; la acción Standard del sistema consiste en listar un comentario y provocar la condición ERROR.

TRANSMIT (NOMBREDEARCHIVO)

Esta condición tiene lugar al detectarse en el archivo especificado un error permanente de transmisión; la acción Standard del sistema consiste en listar un comentario y provocar la condición ERROR.

UNDEFINEDFILE (NOMBREDEARCHIVO)

Esta condición tiene Jugar cuando un archivo denominado en una sentencia OPEN no puede abrirse; la acción Standard del sistema consiste en listar un comentario y provocar la condición ERROR.

UNDERFLOW

Esta condición tiene lugar cuando el exponente de un número de punto flotante es menor que el mínimo permitido definido por la aplicación; el resultado se coloca al menor valor positivo no nulo; la acción Standard del sistema consiste en listar un comentario y proseguir el proceso.

ZERODIVIDE

Esta condición tiene lugar al intentar dividir por cero; no debe hacerse hipótesis alguna sobre el resultado de la división; la acción Standard del sistema consiste en listar un comentario y provocar la condición ERROR.

ASIGNACIÓN DE MEMORIA

Debido a que la memoria interna de un ordenador es limitada en tamaño, frecuentemente resulta de crucial consideración el uso eficiente de esta memoria durante la ejecución de un programa. El proceso estático de asignación de datos utilizado por muchos compiladores, es decir, la asignaci6n de una zona de memoria precisa para los valores de cada nombre de datos que se utiliza en un programa, puede redundar en desperdicio. El uso múltiple de una zona de memoria para Ítems de datos diferentes durante la ejecución de un programa puede reducir la cantidad total de memoria que necesita dicho programa.

El PL/I proporciona diversos métodos de controlar la asignación de memoria a los valores de un determinado nombre de datos del programa. No obstante, no es necesario emplear estos métodos cuando la reducción de memoria sea de escasa importancia.

La asignación de memoria a los valores de un nombre de datos puede tener lugar estáticamente, es decir, antes de la ejecución del programa, o dinámicamente, es decir, durante dicha ejecuci6n. Puede asignarse memoria dinámicamente a los valores de un nombre de datos, y liberarse subsiguientemente dicha asignación. Cuando queda liberada, la memoria está disponible para asignaciones posteriores.

Cada nombre de datos de un programa se describe con una clase de memoria, que especifica la manera de asignar memoria a ese nombre de datos. Existen tres clases de memoria, cada una se especifica declarando el nombre de datos con uno de estos tres atributos de clase de memoria: STATIC, AUTOMATIC o CONTROLLED. Los atributos de clase de memoria pueden declararse también para conjuntos dimensionados y para estructuras principales. Una clase de memoria se aplica a todos los Ítems elementales del conjunto o estructura. La clase de memoria de un nombre de datos puede describirse contextualmente o por omisión.

MEMORIA ESTÁTICA

La memoria de un nombre de datos con el atributo STATIC se asigna antes de la ejecuci6n del programa y nunca se libera durante dicha ejecución. Un nombre de datos declarado con el atributo STATIC puede declararse con los atributos de ámbito EXTERNAL o INTERNAL (estudiados en páginas anteriores de este capítulo). Un nombre de datos EXTERNAL con una clase de memoria no especificada tiene por omisión, el atributo STATIC.

MEMORIA AUTOMÁTICA

Si un nombre de datos tiene el atributo AUTOMATIC, el bloque en que se declara este nombre de datos determina la asignación dinámica para dicho nombre. Al activarse este bloque durante la ejecución del programa se asigna la memoria al nombre de datos, la cual permanece asignada hasta la rescisión del bloque, en cuyo momento la memoria del nombre de datos queda liberada. Obsérvese que la rescisión de un bloque mediante una sentencia GO TO puede motivar la rescisión simultánea de otros bloques y, consecuentemente, la liberación simultánea de la memoria de todos los nombres de datos declarados en estos bloques con el atributo AUTOMATIC.

El atributo de ámbito de un nombre de datos declarado con el atributo AUTOMATJC debe ser lNTERNAL. El atributo INTERNAL puede declararse explícitamente o aplicarse por omisión. Un nombre de datos declarado con el atributo INTERNAL pero con una clase de memoria no especificada tiene por omisión, el atributo AUTOMATIC. Si para un nombre de datos no se especifican los dos atributos de memoria y de ámbito, se le supone aplicada memoria AUTOMATIC.

MEMORIA CONTROLADA

Cuando un nombre de datos tiene el atributo CONTROLLED, la asignación de memoria para dicho nombre de datos debe especificarse con las sentencias ALLOCATE y FREE. Los nombres de datos CONTROLLED pueden declararse con el atributo de ámbito EXTERNAL o INTERNAL.

SENTENCIA ALLOCATE

Al ejecutarse la sentencia ALLOCATE, origina la asignación de memoria al nombre de datos que se especifique declarado con el atributo CONTROLLED. La sentencia ALLOCATE puede tener el formato siguiente:

```
ALLOCATE nombre de datos átributo 1 ... atributo n;
```

El nombre de datos de una sentencia ALLOCATE, puede ser el de un conjunto dimensionado o el de una estructura principal. Si debe asignarse memoria a una estructura principal, debe incluirse en la sentencia ALLOCATE la estructura completa, incluyendo todos los números de nivel e identificadores de la misma forma en que aparecen en la sentencia DECLARE. La sentencia ALLOCATE puede asignar a una estructura una cantidad de memoria que difiera de la cantidad de memoria especificada para la estructura en una sentencia DECLARE. Cuando esto ocurre sólo se necesitan en la sentencia ALLOCATE los atributos de aquellos ítems elementales de la estructura que requieren un nuevo tamaño.

Los atributos especificados con un nombre de datos en una sentencia ALLOCATE son optativos. Cuando están presentes los atributos no pueden contradecir a los especificados para el nombre de datos en la sentencia DECLARE asociada ni a los aplicados al nombre de datos por omisión, con una excepción: la longitud de una serie o el número de ítems de la dimensión de un conjunto pueden diferir de la longitud o número establecido en una sentencia DECLARE. Esto permite que la cantidad de memoria asociada con un nombre de datos varíe durante la ejecución de un programa. Obsérvese, no obstante, que el número de dimensiones especificado en una sentencia ALLOCATE para un conjunto dimensionado no puede diferir del número de dimensiones especificado para el mismo en una sentencia DECLARE

Los únicos nombres de atributos permitidos en una sentencia ALLOCATE son BIT, CHARACTER e INITIAL. Estos atributos solo se necesitan para indicar un cambio en la longitud de una serie o para especificar que va a asignarse un nuevo valor inicial cuando tenga lugar la asignación; de otra forma, no se necesita ningún atributo. Debido a que los atributos FIXED, FLOAT, DECIMAL Y BINARY no están permitidos en una sentencia ALLOCATE, no es posible alterar el tamaño de la memoria asignada a ítems de datos descritos con esos atributos.

Cuando en una sentencia ALLOCATE se especifica la longitud de una serie o el número de Ítems de una dimensión de un conjunto dimensionado, esta información tiene prioridad sobre la dada en una sentencia DECLARE. Si una sentencia ALLOCATE no se especifica la longitud de una serle o el número de ítems de una de las dimensiones de un conjunto dimensionado, debe especificarse en una sentencia DECLARE.

Pueden utilizarse asteriscos en lugar de la longitud de la serie o el número de ítems de una dimensión de un nombre de datos especificado en una sentencia ALLOCATE. Los asteriscos indican que la longitud de la serie o los límites de la dimensión son los mismos que los de la asignación más reciente para ese nombre de datos.

Considérese el ejemplo siguiente:

```
A: PROCEDURE;
    DECLARE PRECIO (100) FIXED (4,2) CONTROLLED,
            CUENTA FIXED (2) INITIAL (50);
             •
             •
             •
        ALLOCATE PRECIO;
             •
             •
             •
             •
         ALLOCATE PRECIO (75);
             •
             •
             •
         ALLOCATE PRECIO (*);
             •
             •
             •
         ALLOCATE PRECIO (CUENTA);
             •
             •
             •
END  A;
```

Se declara que el conjunto dimensionado denominado PRECIO consta de 100 posiciones. Sin embargo, el atributo CONTROLLED indica que la memoria para PRECIO solamente se asignará al ejecutarse una sentencia ALLOCATE para PRECIO. La primera sentencia ALLOCATE en el ejemplo utiliza el número 100 especificado en la sentencia DECLARE como número de posiciones de PRECIO. Al ejecutarse la segunda sentencia ALLOCATE, se asignan a PRECIO solamente 75 posiciones del conjunto. La tercera sentencia ALLOCATE utiliza el número de posiciones establecido en la asignación más reciente de PRECIO. La última sentencia ALLOCATE utiliza el valor de CUENTA para determinar el número de posiciones que deben asignarse a PRECIO.

La clase de memoria para CUENTA es AUTOMATIC. Cada vez que se activa el procedimiento A se asigna memoria a CUENTA y se atribuye como valor de CUENTA el entero 50. Cuando el control deja el procedimiento A, se libera la memoria asignada a CUENTA; sin embargo, la memoria asignada a PRECIO permanece asignada hasta liberarse mediante una sentencia FREE.

Puede utilizarse una sentencia ALLOCATE para asignar memoria a más de un nombre de datos. Los sucesivos nombres de datos que aparezcan en una sentencia ALLOCATE' se separan por comas. En el ejemplo que sigue:

```
ALLOCATE CATALOGO (50), COSTE,
         CODIGO CHARACTER (lO),
         CANTIDAD;
```

la memoria necesaria para los nombres de datos CATALOGO, COSTE. CODIGO y CANTIDAD se asigna cuando se ha ejecutado la sentencia ALLOCATE .

SENTENCIA FREE

La sentencia FREE produce la liberación de la memoria asignada más recientemente a los nombres de datos especificados. La forma general de una sentencia FREE es:

```
FREE nombre de datos 1, ... ,nombre de datos n;
```

Cada nombre de datos debe tener clase de memoria CONTROLLED pudiendo ser también el nombre de un conjunto dimensionado o de una estructura. Si al ejecutarse la sentencia FREE el nombre de datos especificado no tiene asignada memoria, no tiene lugar acción alguna para este nombre de datos. El ejemplo siguiente muestra la utilización de una sentencia FREE junto con una sentencia ALLOCATE:

```
 A: PROCEDURE;
       DECLARE TARIFA (100) FIXED (4,2) CONTROLLED;
         •
         •
       ALLOCATE TARIFA;
         •
         •
       GET LIST (TARIFA);
         •
         •
       PUT EDIT (TARIFA) (F(4,2));
         •
         •
       FREE TARIFA;
         •
         •
END A;
```

TARIFA se declara como un conjunto que contiene 100 posiciones, teniendo clase de memoria CONTROLLED. Al ejecutarse la sentencia ALLOCATE, se genera memoria para TARIFA. A continuación, se introducen los datos por lectura en la tabla de TARIFA y se escriben en salida desde ella y, al ejecutarse la sentencia FREE, queda liberada la memoria de TARI A.

Se puede asignar memoria a un nombre de datos en un bloque y liberarse en otro si dicho nombre de datos se ha declarado de forma que su ámbito incluya ambos bloques.

COMPARACIÓN ENTRE PL/I Y COBOL: ESTRUCTURA DEL PROGRAMA

El estudio que sigue compara las formas de estructurar un programa en PL/I y COBOL. Aunque ambos son lenguajes orientados a problemas y emplean la sentencia como elemento básico del programa para el proceso de datos y para la alteración de la secuencia de ejecuci6n de programas, es precisamente en el campo de la estructuración de programas que radica la mayor diferencia entre el PL/I y el COBOL. La siguiente lista indica algunas de las diferencias más significativas de la estructura de un programa en ambos lenguajes.

1. En general, un programa COBOL es equivalente a un procedimiento externo de PL/I. La Data Division y la Environment Division del COBOL corresponden en su mayor parte a la sentencia DECLARE en PL/I. La Procedure Division en COBOL es equivalente a las sentencias ejecutables de un procedimiento en PL/I. La función desempeñada por la ldentification Djvision en COBOL la ejercen, en PL/I, los comentarios.

2. La sentencia ENTER en COBOL es equivalente a la CALL en PL/I, cuando dicha sentencia CALL se utiliza para activar procedimientos externos compilados separadamente. Sin embargo, el concepto de bloques de procedimiento jerarquizados (bloques de procedimiento interno) del PL/I no tiene equivalente en COBOL. Consecuentemente, no es posible definir dentro del mismo programa fuente COBOL subprogramas internos a los que puedan asignarse argumentos.

3. El COBOL no proporciona los recursos equivalentes del PL/I para asignación automática y controlada de memoria.

4. Las opciones de COBOL AT END, INVALID KEY y SIZE ERROR las procura en PL/I la sentencia ON. Sin embargo, el PL/I proporciona una gama más completa de condiciones de interrupción que el CODOL. El efecto de la sentencia ALTER de COBOL se obtiene en PL/I mediante la asignación de una constante considerada como símbolo al nombre de símbolo de una sentencia GO TO.

6. La sentencia PERFORM de COBOL y la DO de PL/I pueden emplearse para fines similares. Sin embargo la sentencia PERFORM se utiliza para controlar bucles "fuera de línea) mientras que la DO lo hace "en línea”.

5

Manipulación de datos

TABLA DE CONTENIDOS

INTRODUCIIÓN

En general el proceso de datos está relacionado con la utilización de los datos disponibles para generar o modificar otros datos. Por ejemplo, se podrán utilizar los datos contenidos en la ficha de control de tiempo de un empleado, a fin, de generar su cheque de pago y actualizar sus haberes anuales a la fecha que estén almacenados en un archivo maestro. Dichas manipulaciones se indican en PL/I por medio de expresiones que emplean nombres de datos, constantes y operadores. Al calcularse una expresión su valor atribuye a un nombre de datos por medio de sentencias de asignación.

El estudio que sigue muestra cómo pueden emplearse en sentencias de asignación los diferentes tipos de expresiones, para generar o modificar ítems de datos aritméticos y de series. En este capítulo se incluye un estudio sobre especificaciones de modelos y la forma de utilizarlas para modificar los datos mediante la compaginación.

SENTENCIA DE ASIGNACIÓN

La forma de una sentencia de asignación puede escribirse:

```
nombre de datos = expresión;
```

Al principio de una sentencia de asignación no aparece palapra clave alguna. Es el signo igual el que desempeña la función de palabra clave, estableciendo que el valor de la expresión a su derecha va a atribuirse como valor del nombre de datos de su izquierda. Las expresiones de las sentencias de asignación pueden constar de:

- Una constante.
- Un nombre de datos.
- Una secuencia de constantes, nombres de datos y operadores.

La sentencia de asignación que sigue ilustra el uso de una constante como expresión:

```
PRECIO = 19.95;
```

Al ejecutarse esta sentencia, se genera un ítem de datos cuyo valor es equivalente a la constante 19.95, almacenándose en el área de memoria asociada con PRECIO. Cualquier ítem de datos previamente almacenado en el área de memoria de PRECIO deja de estar disponible, puesto que ha sido sustituido por el nuevo ítem de datos.

La representación del nuevo ítem de datos debe concordar con los atributos de PRECIO, lo cual puede implicar la creación de un ítem de datos cuyas características difieran de las de la constante 19.95.

Por ejemplo, si PRECIO tiene los atributos FIXED DECIMAL (5,2), que indican que el ítem de datos denominado PRECIO es un número decimal de cinco dígitos en punto fijo con dos posiciones decimales, el ítem de datos generado para PRECIO en el ejemplo anterior será equivalente a la constante 019.95; automáticamente se suministra el dígito cero de la izquierda para satisfacer la longitud de cinto dígitos, tal como especifican los atributos de PRECIO.

También es posible que la longitud de una constante numérica exceda a la longitud especificada para PRECIO. Considérese el ejemplo siguiente:

PRECIO = 12345.678;

Si PRECIO tiene los atributos FIXED DECIMAL.(5,2), el ítem de datos generado para PRECIO es equivalente a la constante 345.67; en este caso se truncan dos dígitos de la izquierda y uno de la derecha, para satisfacer los requisitos de longitud de cantidad de posiciones decimales que especifican los atributos de PRECIO.

Además de ajustar automáticamente el tamaño del ítem de datos generado, los sentencias de asignación pueden realizar ciertos tipos de ajuste de datos más complicados, que se verán a lo largo del capítulo.

En una sentencia de asignación pueden utilizarse datos de serie.

Considérese el ejemplo que sigue, donde

```
NOMBRE CHARACTER (5);
NOMBRE = 'TOMAS';
```

Al ejecutarse esta sentencia, se genera un ítem de datos equivalente a la constante de serie de caracteres 'TOMAS', almacenándose en el área de memoria asociada a NOMBRE.

Si la longitud de la constante de serie de caracteres excede a la longitud especificada por los atributos de NOMBRE, los caracteres que se suprimen en la serie creada para NOMBRE son los del lado derecho.

Considérese la sentencia:

NOMBRE = 'SEBASTIAN';

Si NOMBRE conserva el atributo CHARACTER (5) la ejecución de esta sentencia genera entonces la constante de serie de caracteres 'SEBAS', asignándola a NOMBRE.

Si la longitud de la constante de serie de caracteres fuera menor que la longitud especificada para NOMBRE, la serie de caracteres generada se rellena por la derecha con blancos hasta cubrir la longitud especificada.

En caso de series de caracteres de longitud variable (VARYING CHARACTER(30);), no se añaden blancos por la derecha. La longitud puede incrementarse o disminuirse mediante subsiguientes sentencias de asignación. Sin embargo, cuando la longitud de la serie de caracteres generada exceda a la longitud máxima de 30 caracteres, se suprimen por el lado derecho de la serie generada los caracteres excedentes.

Las constantes de series de bits se tratan como constantes de serie de caracteres, excepto que en las series de bits que resulten cortas se rellenan por la derecha con bits cero. La supresión de bits en dichas series tiene lugar también por la derecha.

El ejemplo que sigue ilustra lautilizaci6n de una constante de serie de bits en una sentencia de asignación

```
CODIGO BIT(10);
CODIGO = '10101'B;
```

la serie de bits que se construye para CODIGO es equivalente a la constante de serie de bits '1010100000'B.

En PL/I también se permite que una constante considerada como símbolo aparezca como expresión en una sentencia de asignación. Cuando ocurre esto, el nombre del lado izquierdo del signo igual en la sentencia de asignación debe ser un nombre de símbolo.

Supóngase que se ha declarado el nombre INDICADOR con el atributo LABEL, y considérese el uso de INDICADOR en la siguiente sentencia de asignación:

```
INDICADOR = DEDUCCION;
```

Puesto que INDICADOR es un nombre de símbolo, DEDUCCION debe ser un símbolo de sentencia. Al ejecutarse esta sentencia, el ítem de datos generado para INDICADOR es el símbolo de sentencia DEDUCCION.

Existen dos restricciones en la utilización de una constante considerada como símbolo en sentencias de asignación.

- La constante de símbolo no puede ser un nombre de Inscripción, es decir, no puede ser el símbolo de una sentencia PROCEDURE. Esta restricci6n evita la transferencia errónea del control en una sentencia PROCEDURE mediante una sentencia GO TO (las sentencias PROCEDURE deben activarse solamente mediante sentencias CALL)
- La segunda restricción estriba en que el ámbito de la constante considerada como símbolo debe ser el mismo que el ámbito del nombre de símbolo del lado izquierdo de la sentencia de asignación. Sin esta restricci6n, una sentencia GO TO podría erróneamente intentar enviar el control a un bloque exterior al ámbito del nombre de símbolo.

Una expresión considerada en una sentencia de asignación' puede ser un nombre de datos antes que una constante.

```
PRECIO = COSTE;
```

Al ejecutarse esta sentencia, se genera un ítem de datos de valor equivalente al ítem de datos actualmente asignado a COSTE, almacenándose en el área de memoria asociada a PRECIO. El ítem de datos asociado a COSTE permanece inalterado. Las técnicas de conversión estudiadas para constantes se aplican también a esta sentencia.

CONVERSIÓN ENTRE TIPOS DE DATOS

Se pueden utilizar en sentencias de asignación nombres de datos y expresiones con diferentes tipos de datos.

Por . ejemplo si se declara CODIGO con el atributo CHARACTER (10) y MASCARA con el atributo BIT (10), la siguiente sentencia de asignación:

```
CODIGO = MASCARA;
```

Generará para CODIGO una serie de 10 caracteres que contiene solamente los caracteres 1 y 0. Los bits 1 de MASCARA se identifican con los caracteres 1 de CODIOO, y los bits 0 de MASCARA se identifican con los caracteres 0 de CODIGO.

Si la longitud de caracteres especificada para CODIGO excede a la longitud de bits especificada para MASCARA, el Ítem de datos generado para CODIGO se rellena por la derecha con blancos.

El ejemplo anterior ilustra como una sentencia de asignación puede convertir la representación de un valor de serie de bits a su representación equivalente de series de caracteres.

En una sentencia de asignación existen 6 formas de poder convertir valores de datos de una a otra representación:

- Datos de Series de Bits a Datos de Series de Caracteres
- Datos de Series de Caracteres a Datos de Series de Bits.
- Datos de Series de Bits a Datos Aritméticos.
- Datos de Series de Caracteres a Datos Aritméticos.
- Datos Aritméticos a Datos de Series de Caracteres.
- Datos Aritméticos a Datos de Series de Bits.

El estudio que sigue establece las reglas empleadas en PL/l para cada una de estos conversiones.

CONVERSIÓN DE SERIES DE BITS A SERIES DE CARACTERES

Se genera una serie de caracteres que contiene los caracteres 1 y 0. Los caracteres 1 y 0 de la serie de caracteres corresponden a los bits 1 y 0 de la serie de bits respectivamente.

CONVERSIÓN DE SERIES DE CARACTERES A SERIES DE BITS

Se genera una serie de bits que contiene bits 1 y 0, los cuales corresponden, respectivamente a los caracteres 1 y 0 de la serie de caracteres

En la serie de caracteres no están permitidos caracteres distintos de 1 y 0; de otra forma, tendrá lugar la condición de error CONVERSION (véanse "Condiciones ON" en el Capítulo 4).

CONVERSIÓN DE SERIES DE BITS A DATOS ARITMÉTICOS

El valor de la serie de bits se interpreta como un entero binario de punto fijo sin signo y se representa como un Ítem de datos aritméticos. Este Item de datos tiene los atributos del nombre de datos para el cual se genera (por ejemplo decimal de punto flotante,

binario de punto fijo, etc.). Al ser nula la longitud de una serie de bits de longitud variable, el valor de dicha serie de bits (y el del ítem de datos aritméticos) es cero.

CONVERSIÓN DE DATOS DE SERIES DE CARACTERES A DATOS ARITMÉTICOS

La serie de caracteres debe tener la forma de una constante aritmética en PL/I, ya que en en caso contrario da lugar a una condición de error CONVERSION (véase la sentencia ON en el Capítulo 4). El valor de esta constante aritmética se utiliza para generar un ítem de datos aritméticos. Los atributos del nombre de datos para el que se genera el ítem de datos aritméticos determinan la representación de dicho Ítem de datos (por ejemplo, decimal de punto fijo, binario de punto flotante, etc.). Al ser nula la longitud de una serie de caracteres de longitud variable, el valor de dicha serie de caracteres (y el del ítem de datos aritméticos) también es cero.

CONVERSIÓN DE DATOS ARITMÉTICOS A DATOS DE SERIES DE CARACTERES

El valor del ítem de datos aritméticos se representa como una serie de caracteres. El formato de esta serie de caracteres viene determinado por las reglas de salida controlada por lista estudiadas en el Capítulo 3.

CONVERSIÓN DE DATOS ARITMÉTICOS A DATOS DE SERIES DE BITS

El valor del ítem de datos aritméticos se representa como un número binario de punto fijo. La parte entera de este número se utiliza entonces para generar el ítem de datos de serie de bits.

EXPRESIONES QUE CONTIENEN OPERADORES

El PL/I proporciona cuatro clases de operadores aritméticos de comparación de bit y de concatenación. Estos operadores pueden combinarse con constantes y nombres de datos para formar expresiones más complicadas que las consideradas previamente en este capítulo. Tales expresiones pueden aparecer, no solamente en sentencias de asignación, sino también en otras sentencias.

Por ejemplo las sentencias IF y DO emplean expresiones con fines de control. En la sentencia DECLARE. pueden aparecer expresiones con los atributos INITIAL, BIT Y CHARACTER. En las sentencias de entrada/salida se usan expresiones con algunos de los ítems de formato de control como factores de repetición optativos antes de los ítems de formato.

También se utilizan expresiones para especificar valores de subíndices en nombres subindicados. En la sentencia, DISPLAY, también puede utilizarse una expresión para generar un mensaje.

EXPRESIONES ARITMÉTICAS

En PL/I se dispone de cinco operadores aritméticos para las expresiones aritméticas:

- + (suma), - (resta) , * (multiplicación), / (división), ** (exponenciación)

En las expresiones aritméticas pueden emplearse paréntesis al objeto de agrupar las constantes y los nombres de datos que aparezcan en la expresión aritmética y alterar el orden según el cual van a ejecutarse las operaciones. Cuando en una expresión no figuran paréntesis se emplea un sistema de prioridad (estudiado más adelante) para controlar la secuencia de operaciones. Los ejemplos que siguen ilustran cómo pueden cambiarse los nombres de datos y las constantes con operadores aritméticos para formar expresiones aritméticas:

```
 BRUTO-NETO                139.99 + TARIFA          NUMERO * PRECIO
144 * 11.50                 256/LONGITUD            (SUMA/CUENTA)** 0.5
+ 50                        - (2 * INDICE)          A + ((B-C)*(D+ E))
((B* * 2)-(4* A * C)) ** 5
```

No se necesitan blancos en ningún lado de los operadores o paréntesis. Los únicos operadores permitidos como prefijos son el + y el -. Los operadores de prefijo se aplican a constantes y nombres de datos que aparecen a la derecha de dicho operador de prefijo; los operadores de unión (de infijo) se aplican a las constantes y nombres de datos en uno u otro lado del operador de unión.

En estos ejemplos, los operadores + y - se utilizan como operadores de prefijo en + 50 y en - (2*INDICE) y como operadores de unión en 139.99+TARIFA y BRUTO-NETO.

Cuando no se utilizan paréntesis, la prioridad de los operadores aritméticos es:

- **, prefijo +, prefijo -, * , /
- Infijo +, infijo -.

CONVERSIÓN DE DATOS ARITMÉTICOS

En una expresión pueden utilizarse operadores aritméticos con nombres de datos constantes que tengan diferentes características de datos.

Cuando el valor de un número decimal y el de un número binario se combinan mediante un operador aritmético, el valor del número decimal se convierte a binario.

Análogamente, el valor de un ítem de datos de un ítem de datos de punto fijo se convierte a punto flotante, excepto cuando el ítem de punto fijo es un exponente positivo, es decir. un número entero positivo que aparece a la derecha de un operador de exponenciación. En el último caso, el valor de punto fijo no se convierte a punto flotante.

Cuando a la izquierda de un operador aritmético aparecen datos en punto flotante, ell resultado de la operación aritmética está en formato de punto flotante y la exactitud del resultado es la mayor de los dos números implicados en la operación.

Los datos de punto fijo a la izquierda de un operador aritmético hacen que el resultado esté en formato de punto fijo.. Sin embargo, si la operación es de exponenciación y el ítem de punto fijo a la derecha no es un entero positivo, se convierten a formato de punto flotante ambos números implicados en la exponenciación, estando el resultado en formato de punto flotante.

La conversión de formato de punto flotante a formato de punto fijo solamente tiene lugar cuando se requiere explícitamente, como en el caso de una sentencia de asignación que tiene un nombre de datos descrito en punto fijo a la izquierda del signo igual. Si este nombre no puede ajustarse con toda exactitud a su valor convertido a punto flotante, tendrá lugar el truncamiento, según los casos, a ambos lados del punto decimal. pudiendo producirse una condición de error SIZE (Sentencia ON en el Capítulo 4).

Después de efectuadas las conversiones, se realiza la operación aritmética. En la evaluación de una expresión se considera automáticamente la exactitud de los resultados aritméticos intermedios producidos durante la misma; sin embargo, cada aplicación de PL/I puede restringir la exactitud de dichos resultados intermedios al máximo que se especifique.

Se permiten expresiones aritméticas como subíndices (véase Capítulo 2). Al evaluarse una expresión de subíndice el resultado se convierte a binario de punto fijo y se trunca a un valor entero, en caso de ser necesarias tales acciones. Para cada aplicación de PL/I se especifica un valor máximo de los subíndices.

También pueden emplearse en operaciones aritméticas datos de serie; en tal caso, los datos de serie se convierten a datos aritméticos. La conversión satisface las reglas estudiadas previamente en este capítulo, en "Conversión entre Tipos de Datos".

La utilización de datos de serie en expresiones aritméticas puede dar como resultado la ejecución poco eficiente de un programa compilado; esto es especialmente cierto cuando la misma expresión tiene que evaluarse muchas veces durante la ejecución de un programa. Si se desea eficiencia, los datos de serie deben convertirse a su representación aritmética antes de emplearse en expresiones aritméticas.

EXPRESIONES DE COMPARACIÓN

Los operadores pueden utilizarse para formar tres tipos de expresiones de comparación: aritmética, de carácter y de bit.

- Los valores de los ítems de datos aritméticos se comparan algebraicamente; la comparación de valores negativos da resultados menores que la de valores positivos.
- Las series de caracteres se comparan carácter a carácter de izquierda a derecha. Cuando las series de caracteres difieren en longitud, se igualan ambas longitudes por adición de blancos a la derecha de la serie más corta.
- Las series de bits implican la comparación de dígitos binarios de izquierda a derecha. Cuando las series de bits difieren en longitud, la serie más corta se rellena por la derecha con bits cero. La comparación de un bit cero da resultados menores que la de un bit uno.

El valor de una expresión de comparación sé representa mediante una serie de un bit de longitud:

- La constante de bit ‘1’B representa el valor de una comparación verdadera;
- la constante de bit '0'B representa el valor de una comparación falsa.

Los nombres de datos y las constantes utilizados en expresiones de comparación pueden ser de diferentes tipos de datos. Se utiliza una prioridad en los tipos de datos para convertir los valores implicados en expresiones de comparación de uno a otro tipo.

- Los datos aritméticos tienen la prioridad mayor;
- los de serie de caracteres, la siguiente;
- y los de serie de bits la prioridad menor.

La conversión de los tipos de datos en expresiones de comparación siempre va desde la prioridad más baja a la más alta, y sigue las reglas estudiadas previamente en este capítulo.

Se pueden emplear expresiones de comparación en

- Sentencias IF
- Y en sentencias de asignación: CODIGO = HORAS > 40; el bit resultante se aplica a CODIGO, y si es necesario habrá conversión de datos (como ya se ha explicado).

- En expresiones aritméticas: SALARIO = BASE +(HORAS > 40) * BONUS; el bit resultante de la comparación '1' o '0' , se multiplica por BONUS y el resultado se suma a BASE.

Los operadores de comparación tienen una prioridad más baja que los operadores aritméticos. Por tanto, la expresión que sigue:

- BASE +HORAS> 40 * BONUS;

es equivalente a la expresión:

- (BASE + HORAS) > (40 * BONUS)

EXPRESIONES DE SERIES DE BITS

Los tres operadores de serie de bits son:

- ¬ (NO)
- & (Y)
- | (O)

Estos operadores se utilizan con datos de serie de bits para formar expresiones de serie de bits.

El operador & y | son operadores de unión (infijos), es decir, operan sobre una pareja de series de bits. El operador ¬ es un operador de prefijo y consecuentemente opera sobre una sola serie de bits.

Cuando los operadores & y | se aplican a series de bits de longitud desigual, las longitudes se igualan por la adición de bits cero a la derecha de la serie de bits más corta. El resultado de una operación de serie de bits es una serie de bits de igual longitud que las de las series de bits implicadas en la operación.

Las operaciones de serie de bits se realizan bit a bit, de izquierda a derecha. Cada posición de la serie de bits resultante tiene el valor definido en la tabla siguiente:

Si A es	Si B es	¬A es	¬B es	A&B	A|B
1	1	0	0	1	1
1	0	0	1	0	1
0	1	1	0	0	1
0	0	1	1	0	0

Las expresiones de serie de bits que emplean dos o más operadores de serie de bits se evalúan de izquierda a derecha de acuerdo con cierta prioridad sobre dichos operadores. El operador ¬ tiene la prioridad más alta; el operador | tiene la prioridad más baja. Como en el caso de las expresiones aritméticas en las expresiones de serie de bits pueden utilizarse paréntesis para alterar la prioridad de los operadores de serie de bits.

En una sentencia IF pueden utilizarse expresiones de serie de bits, así como expresiones de comparación. En dichas sentencias IF, s, al evaluarse una expresión de serie de bits, la serie resultante contiene uno o más bits 1, la expresión se considera verdadera; en caso contrario se la considera falsa.

En el ejemplo siguiente:

```
IF ¬ (A&B) THEN GO TO NUEVO_PEDIDO;
ELSE GO TO INVENTARIO;
```

cuando la serie de bits resultante de la evaluación de la expresión de serie de bits ¬(A &B) contiene uno o más bits 1, se ejecuta la sentencia GO TO NUEVO_PEDIOO, en

caso contrario se realiza la sentencia GO TO INVENTARIO.

EXPRESIONES DE CONCATENACIÓN

El operador de concatenaci6n || permite que se añada una serie de bits o de caracteres a la derecha de otra.

Cuando en una operación de concatenación uno de los ítems de datos no es una serie de caracteres, dicho ítem de datos se convierte a serie de caracteres y el resultado es una serie de caracteres.

El operador de concatenación puede aparecer con expresiones aritméticas, de comparación y de serie de bits; cuando se emplea de esta forma, este operador de concatenación tiene la prioridad más baja de todos los operadores.

Considérese la sentencia siguiente:

```
COMENTARIO = SALARIO ES $' || BASE + COMISION;
```

Se evalúa la expresión aritmética BASE + COMISION y el resultado se convierte a serie de caracteres. Esta serie se añade entonces a la constante de serie de caracteres ‘SALARIO ES $' y el resultado se asigna a COMENTARIO.

EXPRESIONES DE CONJUNTOS DIMENSIONADOS

Los nombres de datos que se permiten en sentencias de asignación pueden ser nombres subindicados. Considérese la declaración siguiente, donde TARIFA, TASABLE y TASA son los nombres de tres conjuntos dimensionados:

```
DECLARE TARIFA (100) FIXED (4,2),
        TASABLE (100) FIXED (5,2),
        TASA (1OO) FIXED (2,2);
```

La sentencia de asignación que sigue:

```
TARIFA (1) TASABLE (l)* TASA (1);
```

multiplica el primer valor del conjunto TASABLE por el primer valor del conjunto TASA y almacena el resultado en la primera: posición del conjunto TARIFA. Si tienen que realizarse cálculos similares para todas las posiciones de estos conjuntos, puede emplearse una sentencia DO como sigue:

```
DO I = 1 TO 100 BY 1;
    TARIFA (I)= TASABLE (I) * TASA (I);
END;
```

Al ejecutarse este grupo de sentencias el nombre de datos I adopta inicialmente el valor 1, ejecutándose la sentencia de asignación. Después se devuelve el control a la sentencia DO; el valor de 1 se incrementa a 2, ejecutándose de nuevo la sentencia de asignación. Este ciclo de operaciones se repite hasta que I se haya incrementado a 100.

El PL/I proporciona un modo más simple de obtenerlos resultados descritos anteriormente; se puede usar un nombre de conjunto dimensionado sin subíndice para indicar que todas las posiciones del conjunto van a ser objeto de la operación. Por ejemplo la sentencia de asignación siguiente:

```
TARIFA = TASADLE * TASA;
```

es equivalente de hecho al grupo DO estudiado previamente.

Las operaciones realizadas sobre conjuntos dimensionados se efectúan elemento por elemento; por tanto, todos los conjuntos implicados en una expresi6n de conjuntos deben tener el mismo número de elementos y el mismo número de dimensiones.

El resultado de una expresión de conjuntos dimensionados es a su vez otro conjunto dimensionado.

En una expresión do conjuntos dimensionados se puede emplear con los nombres de conjuntos una constante o nombre de datos, que se aplica a cada posición del conjunto. Por ejemplo, si el conjunto unidimensional llamado CUENTA que consta de seis ítem elementales, tuviera los siguientes valores,

```
5, - 10, 11, -3, 2 y 7
```

la expresión de conjunto - CUENTA arrojaría los valores siguientes:

```
-5, 10, -11, 3, -2, -7
```

y la expresión 3*(-CU E N T A) tendría los siguientes valores:

```
-15,30,- 33,9,-6,-21
```

Si INDICE es otro conjunto unidimensional que consta de seis Ítems elementales con los siguientes valores:

```
2,-10,-10,1,0,40
```

el valor de la expresión de conjunto CUENTA + INDICE será entonces

```
7,-20,1,-2,2,47
```

Análogamente, la expresión C U E N T A * IN D IC E tendría los valores:

```
10,100,-110,-3,0,280
```

EXPRESIONES DE ESTRUCTURAS

En las expresiones, están permitidos también los nombres de estructuras. Todas las estructuras de una sentencia de asignación deben tener idéntica estructuración.

A continuación exponemos distintos ejemplos aclaratorios:

```
1 TOTAL, 2 BRUTO, 2 NETO, 2 TARIFA,
1 ANTIGUO, 2 BRUTO_ANTIGUO, 2 NETO_ANTIGUO, 2 TARIFA_ANTIGUO,
1 NUEVO, 2 BRUTO_ NUEVO, 2 NETO_ NUEVO, 2 TARIFA_NUEVO;
```

La sentencia de asignación:

```
 (TOTAL = ANTIGUO *  NUEVO;
```

es equivalente a la siguiente sucesión de sentencias:

```
BRUTO = BRUTO_ANTIGUO + BRUTO_NUEVO;
NETO = NETO_ANTIGUO + NETO_NUEVO;
TARIFA = TARIFA_ANTIGUO + TARIFA NUEVO;
```

En una expresión de estructuras pueden utilizarse, con los nombres de estructuras, constantes y nombres de datos que no son nombres de estructuras o de conjuntos. Cuando se utiliza de esta forma, el valor de la constante o ,nombre de datos se aplica a cada uno de los ítems elementales de la estructura. Utilizando la estructura TOTAL descrita anteriormente, considérese la sentencia siguiente:

```
TOTAL = TOTAL * 2;
```

Esta sentencia es equivalente a la siguiente sucesión de sentencias:

```
BRUTO = BRUTO * 2;
NETO = NETO * 2;
TARIFA = TARIFA * 2;
```

En la misma expresión no pueden figurar nombres de estructuras y nombres de conjuntos dimensionados. Sin embargo, una expresión puede utilizar un conjunto dimensionado compuesto de estructuras. Considérense las siguientes estructuras:

```
TOTAL (1O), 2 BRUTO, 2 NETO, 2 TARIFA,
ANTIGUO (1O), 2 BRUTO, 2 NETO, 2 TARIFA,
NUEVO (10),2 BRUTO, 2 NETO, 2 TARIFA,
```

Cada estructura consta de 30 ítems elementales; los ítems elementales BRUTO, NETO Y TARIFA están repetidos 10 veces en cada estructura. La sentencia: '

```
TOTAL (I) = ANTIGUO (J) + NUEVO (K);
```

es equivalente a la siguiente sucesión de sentencias:

```
TOTAL(I).BRUTO = ANTIGUO(J).BRUTO + NUEVO(K).BRUTO;
TOTAL(I).NETO = ANTIGUO(J).NETO + NUEVO(K).NETO;
TOTAL (I).TARIFA = ANTIGUO(J).TARIFA + NUEVO(K).TARIFA;
```

En estas sentencias se necesita calificación para hacer exclusivos a BRUTO, NETO Y TARIFA. Cada uno de los subíndices I, J Y K especifica uno de los diez grupos posibles de nombres de datos que constan de BRUTO ,NETO y TARIFA.

Puesto que los subíndices pueden trasladarse al lado derecho de un nombre calificado, es posible escribir las sentencias anteriores así:

```
TOTAL.BRUTO(I) = ANTIGUO.BRUTO(J)+ NUEVO.BRUTO(K);
TOTAL.NETO(I) = ANTIGUO.NETO (J)+ NUEVO.NETO (K);
TOTAL.TARIFA(I) = ANTIGUO.TARIFA(J)+ NUEVO.TARIFA (K);
```

ASIGNACIÓN BY NAME

Las sentencias de asignación que implican estructuras pueden realizar la asignación elemento a elemento sobre la base de nombres correspondientes antes que sobre la base de posiciones correspondientes en las estructuras. Este tipo de asignación se llama asignación mediante nombre y se indica añadiendo BY NAME (con una coma de separación) a la derecha de la expresión de estructura en la sentencia de asignación, de la siguiente forma:

```
nombre-estructura = expresión-estructura, BY NAME;
```

Entonces los nombres de datos, correspondientes deben estar contenidos en sus respectivas estructuras de la siguiente manera: "todos los nombres que puedan calificar a los nombres de datos correspondientes deben ser idénticos", excepto el nombre escrito en la sentencia de asignación. Así, podrían utilizarse las estructuras siguientes en una sentencia de asignación con la opción BY NAME:

```
1 MAESTRO,2NOMBRE, 3 NOMBRE_DE_PILA, 3 PRIMER_APELLIDO, 2 DIRECCION,
1 DETALLE ,2 DIRECCION ,2NOMBRE ,3NOMBRE_DE_PILA,3 SEGUNDO_APELLIDO,
PRIMER_APELLIDO;
```

Los nombres correspondientes en estas estructuras son: NOMBRE, NOMBRE_DE_PILA, PRIMER_APELLIDO y DIRECCION. Cabe aclara que la opción BY NAME "NO" requiere que las estructuras sean del mismo tipo. Por tanto la instrucción

```
MAESTRO = DETALLE, BY NAME;
```

Es equivalente a la siguiente sucesión de sentencias:

```
MAESTRO.NOMBRE.NOMBRE_DE_PILA = DETALLELE.NOMBRE.NOMBRE_DE_PILA;
MAESTRO.NOMBRE.PRIMER_APELLIDO = DETALLE.NOMBRE.PRIMER_APELLIDO;
MAESTRO.DIRECCION = DETALLE.DIRECCION:
```

Las expresiones también se evalúan sobre la base de nombres correspondientes cuando dichas expresiones aparecen en una sentencia de asignación que utilice la opción BY NAME. Considérense las siguientes estructuras:

```
1 TOTAL, 2 BRUTO, 2 NETO, 2 TARIFA,
1 ANTIGUO, 2 FECHA, 2 BRUTO, 2NETO,
1 NUEVO, 2 FECHA,2 NETO, 2 BRUTO,
```

La sentencia de asignación siguiente:

```
TOTAL = ANTIGUO + NUEVO, BY NAME;
```

es equivalente a la siguiente sucesión de sentencias:

```
TOTAL.BRUTO = ANTIGUO.BRUTO + NUEVO.BRUTO;
TOTAL.NETO = ANTIGUO.NETO + NUEVO.NETO;
```

Aunque FECHA figura en ambas estructuras ANTIGUO y NUEVO, no es objeto de suma alguna puesto que este nombre no es parte de la estructura T0TAL.

COMPAGINACIÓN DE DATOS

El estudio que sigue presenta un tipo adicional de manipulación de datos que permite la compaginación detallada de los ítems de datos para fines de impresión.

La compaginación puede describirse como una alteración del formato y/o la puntuación de un ítem de datos de serie de caracteres, realizada generalmente para mejorar la legibilidad, o, como en los cheques de pago, proteger el ítem de datos contra alteraciones no autorizadas.

La compaginación implica adición de caracteres al ítem de datos y/o sustitución de los caracteres que se especifiquen por otros.

Por ejemplo, en una aplicación de nómina. los dígitos que representan el salario de un empleado podrían ser 0015089. Estos dígitos serian mucho más significativos en un cheque de pagos en forma compaginada, tal como $** 150.89; los asteriscos dificultarían también cualquier intento de alterar la cantidad.

La compaginación se describe mediante una especificación de modelo, en el atributo PICTURE (estudiado en el Capítulo 2 o bien en la descripción de formato P (estudiada en el Capítulo 3).

El atributo PICTURE que aparece en una sentencia DECLARE con un nombre de datos determina la forma de compaginar los ítems de datos cuando se asignan dicho nombre de datos.

Los ítems de datos pueden asignarse a nombres de datos mediante la sentencia de asignación, la sentencia GET o la sentencia READ.

La descripción de formato P se utiliza para especificar la compaginación de datos durante la ejecución de una sentencia PUT controlada por edición.

El estudio que viene a continuación describe los caracteres de modelo que pueden utilizarse para compaginar un ítem de datos de serie de caracteres. Los ejemplos sirven para hacer ver el efecto de cada uno de los caracteres de modelo, mostrando su relación con los restantes caracteres de modelo.

PUNTO DECIMAL (.)

El punto decimal solo puede utilizarse una vez en una especificación de modelo de datos numéricos. Indica que la posición correspondiente en la serie de caracteres contiene un punto decimal.

SIGNO +

El carácter de modelo + se utiliza de la misma forma que el carácter de modelo $, excepto en que el signo + aparecerá cuando el valor numérico de la serie de caracteres sea mayor o igual que cero; en caso contrario no aparecerá Signo alguno.

DÓLAR $

Cuando en una especificación de modelo se utiliza el carácter $ una sola vez, este debe aparecer en el extremo derecho o bien en el izquierdo de la especificación de modelo. Indica la presencia de un' $ en la posici6n correspondiente de la serie de caracteres. En el extremo izquierdo de la especificación de modelo puede aparece runa secuencia de caracteres $ para indicar un signo de $ flotante, lo cual especifica la sustitución por blancos de los ceros no significativos y por un $ la del cero no significativo situado más a la derecha. Cuando todas las posiciones numéricas de la especificación de modelo contienen el carácter de modelo $ y el valor numérico de la serie de caracteres es cero, todas sus posiciones se cambian a blancos, incluso si existe un punto decimal.

ASTERISCO *

El carácter de modelo * es similar al carácter de modelo Z. salvo que el carácter * se utiliza en lugar de un blanco para sustituir a un cero no significativo. El carácter de modelo * no debe utilizarse con el carácter Z en la misma especificación de modelo

MENOS –

El carácter de modelo es similar al carácter + excepto en que aparecerá un signo - cuando el valor numérico de la serie de caracteres sea menor que cero.

/

El carácter de modelo / se utiliza de la misma forma que la coma, salvo que aparece una barra en la posición correspondiente de la serie de caracteres.

,

El carácter de modelo coma especifica la aparición de una coma en la posición correspondiente de la serie de caracteres. Si tiene lugar la supresión de ceros, la coma solo aparecerá si existe un dígito no suprimido a la izquierda de la coma. Si se han suprimido todos los dígitos a la izquierda de la coma, se presentan tres posibilidades:

- El carácter precedente es *; la posición de la coma contendrá entonces un asterisco.
- El carácter precedente es un $, +, -, o S flotante; la posición de la coma se trata entonces como si contuviera igualmente el carácter flotante.
- El carácter precedente es distinto de los anteriores; la posición correspondiente en los datos contiene entonces un blanco.

B

El carácter de modelo B especifica la inserción de un blanco en la posición correspondiente de la serie de caracteres

CR

La pareja de caracteres CR solamente puede aparecer en el extremo derecho de la especificación de modelo. Indica que la serie de caracteres contiene CR si el valor numérico de dicha serie es menor que cero, en caso contrario, se utilizan caracteres en blanco en lugar de los caracteres CR.

DB

La pareja de caracteres DB se utiliza en la misma forma que la pareja CR, excepto que, cuando el valor numérico de la serie de caracteres es menor que cero, aparecen los caracteres DB

S

El carácter de modelo S es similar a los caracteres + y -, y especifica que siempre aparecerá en la serie de caracteres un signo + o -.

V

El carácter de modelo V solamente se utiliza una vez en la especificación de modelo de datos numéricos. Especifica que debe suponerse un punto decimal en la posición correspondiente de la serie de caracteres, en la que realmente no está presente punto decimal alguno. Se alinea con el punto decimal que haya en el ítem de datos.

Y

El carácter de modelo Y especifica una posición de dígito condicional. Cuando la posición correspondiente en la serie de caracteres contenga un cero significativo o no significativo, se sustituirá el cero por un blanco; en caso contrario quedará un dígito no nulo.

Z

El carácter de modelo Z especifica una posición de dígito condicional. Cuando la posición correspondiente en la serie de caracteres contiene 'un' cero no significativo se sustituirá el cero por un blanco, en caso contrario, quedará un dígito no nulo. Cuando todos los dígitos de la serie de caracteres se sustituyan por blancos también se sustituirá por dicho carácter blanco el punto decimal, si existe. En una especificación de modelo, el carácter de modelo Z no debe utilizarse a la derecha del carácter 9 o a la derecha de un carácter flotante.

9

El carácter de modelo 9 indica que la posición correspondiente en la serie de caracteres siempre contiene un dígito decimal (de 0 a 9).

COMPARACIÓN ENTRE PL/I Y COBOL: MANIPULACIÓN DE DATOS

El estudio que sigue compara los medios de manipulación de datos en PL/I y COBOL. Ambos lenguajes utilizan expresiones para especificar cálculos de datos y la especificación de modelo para compaginar datos. Sin embargo, el PL/I utiliza una sola sentencia para todos los tipos de manipulación de datos, mientras que el COBOL emplea varias sentencias diferentes. La lista que sigue contiene algunas de las diferencias más significativas entre los medios de manipulación de datos de ambos lenguajes:

1. Los efectos de las sentencias de COBOL, MOVE, COMPUTE, ADD, SUBTRACT. MULTIPLY y DIVIDE se consiguen en PL/I con la sentencia de asignación. Sin embargo, al emplearse en COBOL la sentencia MOVE con grupos (equivalente a las estructuras de PL/I) los datos se mueven sin tener en cuenta la estructura de nivel de los grupos, ignorándose la conversión de los datos, si se especifica. Cuando en PL/I se aplica la sentencia de asignación a estructuras, la asignación se realiza ítem elemental a ítem elemental, efectuándose las conversiones de todos los datos según se haya especificado.
2. La opción de PL/I BY NAME es similar a la opción CORRESPONDING en COBOL.
3. El PL/I permite que las expresiones utilicen ítems de datos que contienen caracteres de compaginación. El COBOL no dispone de este recurso.
4. Los operadores de serie de bits en PL/I son similares a los operadores lógicos en COBOL. Sin embargo, el COBOL no tiene ningún tipo de datos que corresponda al tipo de datos de serie de bits de PL/I.
5. El operador de concatenación de PL/I no está disponible en COBOL.