COBOL

lenguaje de programación

PRIMERA PARTE

INTRODUCCION INFORMAL DEL LENGUAJE

Redactados por

J. MARIN CORREA
C. NORES GONZALEZ
R. PEÑA MIMO
J. RODRIGUEZ DOMINGO
J. L. SALILLAS IBAÑEZ

Profesores del Instituto de Informática

Depósito Legal M. 947 - 1971
Imprime: R. García Blanco - Avda. Pedro Díez, 3 - Madrid

El presente texto de lecciones de lenguaje de programación COBOL pretende conseguir, con un mínimo de extensión, objetivos difícilmente armonizables: un conocimiento bastante amplio del lenguaje mediante una exposición orientada hacia una mayor eficacia didáctica.

El esfuerzo desplegado por José Luis Salillas Ibáñez, profesor coordinador de la asignatura durante el curso 1969-1970, ha sido el principal factor en el logro de este texto. El Instituto de Informática confía en su eficacia como vehículo de introducción del alumno al dominio del lenguaje COBOL.

INDICE

TEMA I

INTRODUCCION AL LENGUAJE

1.- Necesidad de un lenguaje para tratar la información administrativa y comercial

Los procesos de tipo administrativo y comercial, que llamaremos en general de gestión, tienen características especiales que les distingue claramente de otros procesos como son los cálculos ingenieriles, científicos, lingüísticos, etc. Así, mientras el cálculo aritmético que requieren es por lo general extremadamente simple, en cambio las decisiones lógicas a tomar son, incluso en casos elementales, sumamente complejas.

El proceso de un cálculo científico se puede justificar por una fórmula matemática empleando un simbolismo universalmente aceptado y conocido, de una forma breve y esquemática. Utilizando el simbolismo de las matemáticas se pueden describir brevemente los más complicados algoritmos.

La gestión no ha alcanzado todavía este nivel de abstracción, y aunque utilizando notaciones de conjuntos, grafos y lógica simbólica existen hoy revistas y libros técnicos que tratan al estilo matemático los problemas de gestión, es en cambio una realidad de que carecen de universalidad en el uso, y aplicabilidad en la mayoría de los procesos de cálculo electrónico.

Precisamente una de las características esenciales de los procesos de gestión es la documentación. Esto es lo opuesto a la abstracción, en gestión es preciso detallar con toda claridad cientos de particularidades de la aplicación: descripción de los registros, composición

de los archivos, clave de los datos, etc. En los procesos de tipo matemático el uso, aunque frecuente, de archivos es secundario, y éstos suelen ser sumamente sencillos de composición: tablas de mediciones climáticas, datos de una estructura geométrica, etc., que no contienen nada más què media docena de tipos de variables cuando mucho. En cambio piénsese en la prolijidad de datos que contiene un registro de empleado (nombre, familia, D.N.I., conceptos de sueldos, de descuentos, de anticipos, etc.), de los materiales de un almacén, de los habitantes de un país, etc.

Esta variedad de elementos que intervienen en una aplicación de gestión exige una documentación exhaustiva. Si a esto se añade el que los programas de gestión son de larga duración, pero confrecuentes revisiones, se llega a la conclusión inmediata de que su redacción debe ser clara y adaptable.

A un programa FORTRAN por ejemplo, se le puede hacer claro documentándole con numerosos comentarios, además de otra documentación en forma de organigramas, tablas de decisión, etc., que pueda contener un informe. Entonces si hay que efectuar un cambio en el programa, hay que acompañarle de los correspondientes comentarios. Si el lenguaje que se utilizare fuese lo suficientemente descriptivo, esto sería innecesario.

Otra característica fundamental es la frecuencia con que exigen la impresión de informes sumamente detallados con encabezamientos, totales, etc. Y es bien conocida (por los programadores) que programar el formato de impresión en FORTRAN, ALGOL o en lenguajes ensambladores, suele ser una parte del programa tan costosa o más (en tiempo) que el programa de cálculo.

Se dice que el COBOL no sería el COBOL sin la edición. Aunque la descripción correcta de lo que significa el verbo editar corresponde a uno de los temas de la asignatura, indicaremos aquí que editar significa que si por ejemplo tenemos almacenado en memoria 12684, debería aparecer en la hoja impresa $ AA 126.84 CR (A = blanco). El incluir puntos y comas, anteponer signos + o −, *, suprimir ceros a la izquierda, incluir blancos como en una ficha 24 A 6 A 69, etc., es de una utilización constante en programas de gestión.

Los alumnos de la asignatura, que simultáneamente cursan la de FORTRAN, no deben de sacar la consecuencia de que todo lo anteci - cho no se pueda programar en FORTRAN. De hecho con el propio lenguaje FORTRAN se puede programar casi la totalidad de lo que se puede

programar en COBOL (lo contrario también es cierto), y utilizando la facilidad de las subrutinas, una instalación particular (y en efecto existen paquetes de subrutinas comerciales) se la puede dotar de las necesarias para poder simular al lenguaje COBOL, incluso en la edición.

Pero aun en este caso extremo, el lenguaje FORTRAN se queda en inferioridad clara para resolver problemas como la modificación de la composición de un registro o la longitud de un campo, problemas que se resuelven en COBOL de forma incomparablemente más simple.

A esto debe añadirse la cualidad básica de que en FORTRAN las palabras (variables) tienen una longitud máxima de cinco o seis caracteres, en cambio en COBOL se pueden formar palabras de longitud seis veces mayor y además se las puede calificar (adjetivar). Esto aumenta la legibilidad de un programa COBOL (con muchas variables) respecto a un programa en FORTRAN. Y si lo más que podemos hacer en FORTRAN para tratar los programas de gestión es simular al COBOL, el uso de este lenguaje queda suficientemente motivado.

2.- Lenguajes de alto nivel

Una vez expuesta la necesidad de un lenguaje de gestión, y mencionado el COBOL como apropiado para estos tratamientos, indicaremos que su puesto, entre los lenguajes de alto nivel, en compañía del FORTRAN, ALGOL, PL/I, SNOBOL, SIMSCRIPT, etc

Como recordatorio diferenciaremos aquí a los lenguajes de alto nivel, en que su traducción a lenguaje máquina requiere un compilador, esto es un programa que traduce por cada instrucción en lenguaje fuente (alto nivel) una o varias en lenguaje máquina. Los lenguajes simbólicos o ensambladores traducen una instrucción fuente por una de máquina, aunque esto tiene frecuentes excepciones en la mayor parte de los modernos ensambladores, en las instrucciones llamadas macro o macroinstrucciones.

La historia comercial de los lenguajes de altonivel parece que se inició en el PACT (1953) (Nota: los datos históricos que siguen están tomados de SPROWLS, COMPUTER PROGRAMMING LANGUAGES), pero su elaboración fue tan costosa que el ordenador para el que se construyó se había quedado obsoleto cuando se puso en servicio. El FORTRAN-I apareció en 1957, y el ALGOL en 1958. El FORTRAN y el ALGOL son lenguajes algorítmicos (para tratamientos matemáticos), siendo el ALGOL el primer esfuerzo internacional para lograr un lenguaje universal.

En el campo de la gestión los lenguajes predecesores del COBOL más importantes fueron el FLOW-MATIC de Sperry Rand UNIVAC (1956), FACT de Minneäpolis Honeywell (1958) y el COMMERCIAL TRANSLATOR de IBM (1959).

3. - Compiladores

Hemos dicho que todos los lenguajes de alto nivel utilizan un programa compilador para traducirlos. Debe quedar clara la diferencia de un lenguaje y un compilador, no debiendo ser motivo de confusión el que a los lenguajes de alto nivel se les llame a veces lenguajes compiladores, haciendo mención en este nombre a la propiedad de ser traducidos por un compilador.

Un compilador es un programa, un programa en lenguaje máquina, ejecutable en un ordenador determinado. Aquí también se utiliza el nombre con relajaciones. Por ejemplo a un paquete de fichas que contiene un programa que cargado en un ordenador y editado es un compilador, se le llama también un compilador. Al programa escrito, cuyo fin es compilar programas escritos en un determinado lenguaje se le llama también compilador. Obsérvese que este programa escrito no tiene que estarlo en lenguaje máquina (generalmente no lo está) sino en un lenguaje ensamblador (lo más común) o en otro lenguaje de alto nivel. Si este lenguaje de alto nivel está orientado a escribir compiladores, entonces se le llama compilador de compiladores.

4. - Descripción de la compilación

Para ejecutar una compilación son necesarios tres elementos fundamentales: un ordenador, un programa compilador y un programa fuente escrito en el lenguaje a traducir. En esta operación el programa fuente (perforado en fichas, cintas, o en banda magnética, etc.) se utiliza como dato del proceso.

Imaginemos el supuesto de que tenemos un ordenador, un compilador perforado en un paquete de fichas, en lenguaje máquina ejecutable, y otro paquete de fichas con el programa fuente escrito en COBOL. La operación de compilación podría ser así:

1. - Colocaríamos el programa traductor en la lectora de fichas.

2.- Utilizando los dispositivos de arrancada automática del ordenador leeríamos estas fichas cargando el programa en memoria.

3.- Como es usual (por cualidades propias de los ordenadores y de los programas en forma ejecutable) cuando termina de almacenarse en memoria, el control pasa al programa. Esto quiere decir que la primera instrucción a ejecutar, es una instrucción del programa traductor que inicia la serie de instrucciones que efectúan la traducción.

4.- Las fichas del programa a traducir estarán colocadas detrás del programa traductor, por lo tanto cuando se ha leído el programa compilador, estas fichas serán las próximas a leer.

5.- El programa compilador irá leyendo (dando órdenes de leer) las fichas del programa fuente una a una.

NOTA.- La forma de efectuar la traducción que sigue es similar simplificado de las muchas y complejas formas de compilar que existen.

6.- En memoria (del ordenador) se analiza el contenido de la ficha, y se genera una serie de claves para relacionar el contenido de esta ficha con las anteriores y las que siguen, además de la información propia e independiente de la ficha.

7.- Cuando se lee la ficha END OF PROGRAM (por ejemplo) del programa fuente, el compilador da por terminada la lectura de fichas y pasa a analizar y cubrir (para direcciones, etc.) las claves que ligan a unos elementos con otros del programa. Entonces es cuando quedan completamente construidas las instrucciones en código máquina (frecuentemente en un lenguaje intermedio: ensamblador).

8.- Por último este programa (objeto) se perfora en fichas, por órdenes dadas por el compilador a la perforadora de fichas acoplada al sistema.

En un sistema como el imaginado, cuando se quiera ejecutar el programa traducido, habría que cargarlo en memoria utilizando las fichas perforadas en lenguaje máquina que nos produjo la compilación. Una vez cargado en memoria pasaría el control a la primera instrucción ejecutada de este programa.

En los sistemas operativos modernos la compilación se ejecuta de forma muy distinta aparentemente, pero en esencia de igual for-

ma. El programa fuente sigue estando (por lo general) perforado en un paquete de fichas. El programa traductor en cambio se encuentra almacenado en un disco del sistema o en una banda magnética. Para que actúe (para que se cargue en memoria y se ejecute) es necesario llamarle. Esto se hace por medio de otro programa (o uno de una serie de programas) llamado monitor.

A este programa monitor se le llama a su vez empleando los dispositivos del ordenador (su localización es fija y por lo tanto conocida por el operador) o por fichas de control (fichas de datos del sistema operativo). Insistiremos en que el sistema operativo consiste en un conjunto de programas, en el que el monitor (otras veces llamado de formas diferentes, como supervisor) actúa de enlace.

Con las adecuadas fichas de control se indica al monitor que se desea que actúe el compilador de COBOL. Se carga éste en memoria y a su vez lee otras fichas de control (estos datos para este programa específicamente) que le indican la forma (de las posibles en el sistema) que se quiere efectuar la compilación. Entonces actúa el compilador y se ejecuta la traducción, pero ésta queda almacenada en una de las memorias masivas del sistema (disco, tambor, banda magnética) de acceso rápido. Obsérvese la diferencia entre sistema de proceso electrónico de datos (hardware) y sistema operativo (software).

Para ejecutar el programa traducido y almacenado se utilizan también las fichas de control del sistema operativo. Como la compilación se efectúa con tal rapidez en los modernos ordenadores, y las fichas de control entremezcladas con las del programa fuente y las de datos, se leen sin ninguna intervención manual, a ojos inexpertos parece que el programa fuente es entendido por el ordenador.

5. - Nota histórica del COBOL

El Algol fue el primer esfuerzo internacional en pro de un lenguaje algorítmico universal. En este empeño se intentó lograr en lenguaje más potente para describir algoritmos y que fuese independiente al tipo de máquina utilizado.

La rápida proliferación de máquinas distintas producidas además por una docena de fabricantes independientes, hacía desear la existencia de lenguajes independientes de la máquina en donde se habrían de ejecutar los programas. Si esto en programas científicos era importante, tanto más lo es en programas de gestión que se ejecutan con regularidad y cuya suspensión puede originar una situación de catástrofe.

El usuario más importante, el gobierno USA, habría de sentir la necesidad más que nadie, por ello el Pentágono convocó una reunión de fabricantes en 28 y 29 de mayo de 1959. En esta reunión fue reconocida la conveniencia y la posibilidad de tal lenguaje. A tal fin se constituyen tres comités denominados Short Range, Intermediate Range y Long Range. En septiembre de 1959 se elevó un informe al Comité Ejecutivo en el que se enunciaban las bases del nuevo lenguaje. El 7 de noviembre de 1959 se aceptó el nombre de COBOL (COMMON BUSINESSORIENTED LANGUAGE). El informe con el lenguaje ultimado se elevó a Comité Ejecutivo de CODASYL (Conference on Data Systems Languages el 17 de diciembre de 1959). En Abril de 1960 fue publicado porla oficina de prensa del gobierno americano.

Se creó un comité de mantenimiento y un comité especial para revisar y poner al día el lenguaje. Fruto de estos trabajos fue la edición a mediados de 1963 del COBOL 1961 Extended.

En noviembre de 1965 el COBOL Committee de CODASYL publicó el COBOL - Edition 1965. También en abril de 1965 el Comité Ejecutivo de CODASYL creó el CODASYL Systems Committee (C S C) con la misión de seguir la investigación del lenguaje.

Por su parte la A S A (American Standard Association) que desde 1960 había intentado elaborar unas normas COBOL se transformó en agosto de 1966 en la United States of America Standards Institute (USASI) siendo el COBOL normalizado un subconjunto del COBOL - Edition 1965.

A pesar de que el empeño de CODASYL y posteriormente de USASI ha sido dirigido a conseguir un lenguaje universal, la realidad es que los COBOL ofrecidos por los fabricantes tienen generalmente peculiaridades propias que le quitan universalidad.

El objeto de nuestro estudio sería un subconjunto del COBOL 1969 suficientemente amplio para abarcar las principales características de los COBOL utilizados en España y que son universales, aunque en aplicaciones prácticas y ejercicios nos veamos obligados a tratar algunas características propias sólo de algún ordenador dondese realicen las prácticas.

6.- Las divisiones del COBOL

Básico para obtener la universalidad de su uso, independiente de la máquina empleada, y para la flexibilidad de las actualizaciones de los programas, así como para su documentación, es la estructura del lenguaje.

Los programas escritos en COBOL se organizan en cuatro divisiones llamadas

IDENTIFICATION DIVISION
ENVIRONMENT DIVISION
DATA DIVISION
PROCEDURE DIVISION

que aparecen en el programa precisamente en este orden. A su vez estas divisiones se dividen en secciones y otras unidades menores que se irán estudiando en los temas que siguen.

El objeto de la IDENTIFICATION DIVISION es documentar la identificación del programa, nombre, autor, fecha, comentario sobre un objeto, etc. Excepto el nombre, el resto de las entradas en esta división no tiene influencia en el compilador. Esto es, no da lugar a que en la traducción se generen instrucciones máquina. El nombre en cambio si puede convertirse en la clave de almacenamiento en la librería del sistema operativo.

La ENVIRONMENT DIVISION sirve para enlazar las unidades físicas que se han de utilizar durante el proceso (operación) con el programa. Es así como se pretende lograr una universalidad del lenguaje, pues en principio debía bastar el cambio de las pocas y breves sentencias que se escriban en esta sección, para que el programa pudiera ejecutarse en un ordenador distinto.

El principio que se usa es el asociar los nombres que se dan en el programa a los elementos físicos que se usan (nombres lógicos), con nombres que en una determinada instalación se dan a los elementos físicos.

Ejemplo:

En un programa utilizamos un archivo llamado CINTA-ENTRADA que consideramos que está grabado en banda magnética, pues bien, debemos relacionar este nombre lógico con la banda que vamos a montar

en un determinado armario (digamos que existen cuatro). El procedimiento usual es identificar este armario. Esto se puede conseguir con el sistema operativo, en el que cuando se instaló (generó) en el sistema de proceso de datos (equipo donde vamos a realizar el trabajo), al armario se le dio la identificación 5 (los otros armarios podrían tener números como 2, 7 y 8).

El sistema operativo nos permite relacionar el número del armario con un nombre, así: TAPE-1 = 5.

Entonces si en la ENVIRONMENT DIVISION escribimos SELECT CINTA-ENTRADA ASSIGN TO TAPE-1 UTILITY 400 el compilador al hacer la traducción, asigna a CINTA-ENTRADA el armario TAPE-1 y cuando el programa se ejecute es el sistema operativo el que por medio de una ficha de control como la anteriormente mencionada, asigna el armario 5 a TAPE-1.

El lector pensará que todo irá bien mientras que la banda que está colocada en el armario 5 sea la que pensamos, pero no en caso de que nos la hayan cambiado. Esto también puede comprobarse utilizando etiquetas que preceden a la información del archivo en banda. Pero esto se estudiará con detalle más adelante.

De forma semejante a la del ejemplo se relacionan todos los nombres lógicos del programa con los elementos físicos empleados; lectoras de fichas, perforadoras, impresoras, etc.

En la DATA DIVISION es donde se describen los formatos y clases de variables y literales que va a utilizar el programa. Es en esta división donde el COBOL cumple su misión de documentar plenamente el programa y al mismo tiempo le hace extremadamente flexible a transformaciones.

Imaginemos que tenemos una variable que es COMISION, que en el programa original hemos definido como capaz de contener cantidades de seis cifras como máximo. En caso de que deseemos aumentar a siete, bastará cambiar su definición en la DATA DIVISION sin tener generalmente mayores complicaciones.

Por último, es en la PROCEDURE DIVISION donde se describen el flujo de operaciones (instrucciones) de que consta el programa.

7.- La hoja de codificación

En la figura 1 aparece una reproducción de la hoja de codificación. Su composición y normas de codificación completa la organización que debe tener un programa escrito en COBOL. La cabecera, compuesta por tres líneas no pertenece al lenguaje y el significado de sus casillas es evidente por sí mismo. Pero señalaremos que en algunos ordenadores la codificación de signos especiales como la Ñ o < no son las perforaciones habituales, entonces en PERFORACIONES debajo del signo correspondiente se indican las perforaciones. Otras veces el motivo puede ser el que la perforadora donde se ha de perforar el programa no contiene tecla para alguno de los signos especiales COBOL.

El campo identificación (73, 80) tampoco lo utiliza el compilador sirve sólo para fijar el contenido de la ficha, generalmente el nombre del programa.

Página (1, 3) y línea (4, 6) tampoco tienen influencia en el programa objeto, pero el compilador comprueba y si existe error lo indica pero no interrumpe la traducción.

Es conveniente numerar así:

```
050    010
050    020
050    030
050    040
 .      .
 .      .
 .      .
```

pues si se desea intercalar un grupo de fichas se procede así:

```
050    020
050    021
050    022
050    023
050    024
050    030
050    040
 .      .
```

La columna 7 se utiliza para indicar que esta línea es continuación de la anterior, esto se indica con un guión (-). Se utiliza p

continuar literales no numéricos que por su longitud frecuentemente no caben en una sola línea.

Fundamental en la codificación de un programa COBOL son los márgenes A y B (8 y 12 respectivamente). Cuando describamos la sintaxis de las divisiones, secciones, párrafos, entradas y sentencias del COBOL nos referiremos constantemente a estos márgenes.

Así los nombres de división se escriben empezando en el margen A

```
A       B
8       12

IDENTIFICATION DIVISION
ENVIRONMENT DIVISION
DATA DIVISION
PROCEDURE DIVISION
```

TEMA II

ELEMENTOS DEL LENGUAJE

1. - Lista de elementos del COBOL

El COBOL como lenguaje que es se compone de partículas elementales que combinadas dan lugar a unidades mayores que a su vez se combinan formando unidades superiores hasta llegar a la unidad suprema que es el programa.

1.1. La partícula más elemental del lenguaje son los caracteres. La A, la B, etc., son caracteres igual que lo es el . o el signo). Cada uno de ellos en un ordenador tiene una determinada representación en bits. Por ejemplo con seis bits se pueden representar 2^6 caracteres (64) y con ocho bits 256.

En un ordenador determinado en que un carácter se representa con seis bits las letras C, D y E podrían tener las representaciones:

C = 100110 = 38
D = 100111 = 39
E = 101000 = 40

A ese valor numérico de su representación en bits (38 para C, 39 para D, etc.) se llama el equivalente entero del carácter. El valor numérico del equivalente entero de cada carácter permite ordenarles, a esta ordenación se la llama secuencia de intercalación. Cada ordenador tiene su propia secuencia de intercalación que el programador debe de conocer pues es fundamental en COBOL para operaciones de comparación y clasificación.

A título ilustrativo damos una secuencia de intercalación de un juego de caracteres COBOL:

b (blanco)
A, B, ..., Z
)
- (guión o signo menos)
+
<
=
>
$
*
(
, (coma)
0, 1, 2, . . . , 9
" (comillas)
; punto y coma
/ barra
. punto o punto decimal

1.2. Con el juego de caracteres se puede construir fácilmente una unidad superior, los literales. Hay literales numéricos y no numéricos.

Los literales numéricos se construyen con los números 0 al 9 con el punto decimal y con el signo + o - que se coloca delante del literal. La longitud máxima de un literal es de 18 cifras estando prohibido colocar el punto decimal como último carácter a la derecha, pues en este caso se confundiría con el punto final de sentencia (unidad de la PROCEDURE).

Ejemplos: 126.64; - 8.6; 41.0

Los literales no numéricos se construyen con todo el juego de caracteres excepto la comilla. A esta secuencia de caracteres se la antepone y se la sigue de una comilla. La longitud máxima es de 120 ó 132 caracteres según los ordenadores. Esta longitud hace que a veces no quepa en una sola línea de la hoja de codificación, para estos casos se utiliza la columna 7 donde se codifica un guión (-) pudiéndose continuar el literal a partir del margen B. (Nota: la convención para continuar literales no numéricos puede variar ligeramente en algunos ordenadores por lo que es necesario consultar el manual particular en cada caso).

Ejemplo:

' ACTUALIZACION RENTAS * * ZONA CENTRO * *'

1.3. Con los caracteres pueden formarse palabras. Las palabras COBOL válidas se construyen con las letras A a la Z, los números 0 al 9 y el guión (-). La longitud de una palabra no puede superar los 30 caracteres y el guión no puede ocupar ni la primera ni la última posición.

Ejemplos:

NOMBRE, 126, A-64

Obsérvese que nunca puede confundirse una palabra con un literal no numérico, puesto que estos siempre están rodeados de comillas. También nótese que el blanco tampoco puede estar dentro de una palabra, (un blanco es un carácter de puntuación, sirve para separar palabras y signos).

Las palabras sirven para

1.3.1. nombres; Ejemplos: IMPORTE, PRECIO, FILLER, FINAL, PICTURE...

1.3.2. verbos; Ejemplos: MOVE, COMPUTE, ADD...

1.3.3. predicados; Ejemplos: COMPUTATIONAL, DISPLAY, RIGHT..

1.3.4. conectias; Ejemplos: AND, THEN, OR ...

1.3.5. relaciones; Ejemplos: GREATER, EQUAL y LESS

1.4. Con los caracteres pueden formarse también cadenas de edición (utilizadas con la cláusula PICTURE). Las reglas de formación de estas cadenas de edición por su complejidad las dejamos para cuando se describa la cláusula PIC (Tema VIII). Sólo diremos que su longitud máxima es de 30 caracteres.

1.5. Con palabras, literales y cadenas de edición se forman las cláusulas. Las cláusulas sirven para definir exactamente las características de los datos.

Ejemplos:

```
USAGE IS DISPLAY
PICTURE 999V99
VALUE IS 'A-63'
```

1.6. Con números, nombres, cláusulas y un punto final se forman las entradas. Las entradas sirven para describir nombres de dato.

1.7. Los registros son descritos por un conjunto de entradas. A su vez su nombre es parte de una entrada independiente.

1.8. Las características de los archivos también se describen con una entrada compuesta con cláusulas especiales.

1.9. La unidad superior que contiene a las entradas de archivos y registros es la sección. En la DATA DIVISION hay cuatro secciones.

FILE SECTION.
REPORT SECTION.
WORKING-STORAGE SECTION.
CONSTANT SECTION.

Nótese que la descripción de datos, registros y archivos pertenece a la DATA DIVISION.

1.10. A su vez en la PROCEDURE DIVISION con nombres, operadores y relaciones se pueden formar expresiones. Hay expresiones aritméticas y expresiones condicionales.

Ejemplos:

```
SUELDO + TRIENIOS
SUELDO > 1000
```

1.11. Con verbos, nombres, conectivas y expresiones, se forman declaraciones (también llamadas instrucciones u órdenes).

Ejemplos:

```
GO TO CALCULO-PRECIO
ADD A TO B GIVING C
```

Las declaraciones pueden ser imperativas como las anteriores o condicionales como:

```
IF A < B GO TO CALCULO-PRECIO ELSE GO TO FINAL.
```

Pueden ser también compuestas: formadas por más de una declaración imperativa o condicional.

1.12. Una declaración simple o compuesta terminada en un punto es una sentencia.

1.13. Un nombre (de párrafo) seguido por un punto y un conjunto de sentencias forman un párrafo.

1.14. La unidad superior al párrafo es la sección, que en la PROCEDURE recibe un nombre dado por el programador.

1.15. La unidad superior a las secciones es la DIVISION. Las divisiones IDENTIFICATION DIVISION y ENVIRONMENT DIVISION también están formadas con párrafos y a los elementos que los forman se llaman a veces cláusulas y más propiamente declaraciones.

2.- Caracteres para formar palabras

Son:

0, 1, .. al 9
A, B a la Z
- (guión)

Las reglas de composición ya han sido dadas.

3. - Signos de puntuación.

' comilla
(paréntesis izquierdo
) paréntesis derecho
b espacio en blanco, a veces se le designa con Δ
. punto
, coma
; punto y coma
' debe colocarse delante y detrás de los literales no-numé - ricos
(debe ir precedido de un blanco
) , . , , y ; deben ir seguidos de un blanco
, y ; se pueden omitir aun cuando están descritos en los formatos sintácticos.

4. - Signos para formar expresiones aritméticas

\+ sumar
\- restar
* multiplicar
/ división
** exponenciación
(paréntesis izquierdo
) paréntesis derecho

El signo = se utiliza con el significado de asignación con el verbo COMPUTE (igual que en FORTRAN).

A los operadores +, -, *, / y ** se les antepone y sigue de blancos (uno al menos).

Distíngase a los operadores + y - de los signos en literales numéricos.

5. - Signos utilizados en expresiones condicionales

Son las relaciones

=

<

>

se les precede y sigue de blancos (al menos uno).

Nótese que el = aquí (una relación) tiene un significado totalmente distinto al = de asignación mencionado antes.

6. - Clases de palabras, reservadas del COBOL y nombres dados por el programador

Hay dos importantes clases de palabras; las reservadas COBOL y los nombres dados por el programador.

Hay una tercera clase de palabras que son nombres con los que se designan unidades, funciones o dispositivos específicos del equipo. Estos nombres son dados por los diseñadores del sistema operativo y por lo tanto para el programador son similares a las palabras reservadas COBOL.

En el cuadro II-a se relacionan las más importantes palabras COBOL reservadas. Estas palabras, descritas con mayúsculas en el metalenguaje, tienen significados especiales y fijos. Cuando se usan hay que escribirlas con su ortografía exacta. Unas son obligatorias en determinadas funciones, mientras que otras son opcionales (sólo sirven para aclarar la lectura del texto). A las primeras se las subraya en el metalenguaje.

7. - Claves de nombres y reglas de formacion

Los hombres que puede dar el programador deben de cumplir las reglas de construcción de las palabras COBOL.

Las clases son:

7.1. Nombres nemotécnicos que se dan en el párrafo SPECIAL-NAMES de la ENVIRONMENT DIVISION.

7.2. Nombres de datos elementales que se dan en la DATA DIVISION.

7.3. Nombres de grupos, también en la DATA

7.4. Nombres de registro, también en la DATA

7.5. Nombres de archivo, también en la DATA

7.6. Nombres de condición, también en la DATA y en SPECIAL - NAMES.

7.7. Nombres de procedimientos en la PROCEDURE DIVISION

7.7.1. Nombres de sección.
7.7.2. Nombres de párrafo.

Los nombres de la DATA DIVISION y ENVIRONMENT DIVISION tienen que tener por lo menos una letra, los de la PROCEDURE pueden estar formados sólo por números.

8. - Literales numéricos y no numéricos

Ya hemos dado sus reglas de formación, son utilizados los numéricos en todas las divisiones pero principalmente unos y otros en la DATA con la cláusula VALUE y en la PROCEDURE con las expresiones.

9. - Constantes figurativas

Con distintos fines se utilizan unas palabras COBOL reservadas que reciben el nombre de constantes figurativas.

Estas son:

ZERO, ZEROS, ZEROES
SPACE, SPACES
HIGH-VALUE, HIGH-VALUES
LOW-VALUE, LOW-VALUES
UPPER-BOUND, UPPER-BOUNDS
LOWER-BOUND, LOWER-BOUNDS
QUOTE
ALL

El nombre de estas constantes es significativo por sí mismo.

La diferencia entre UPPER-BOUND y HIGH-VALUE es que HIGH-VALUE es el máximo o más alto de la secuencia de intercalación,

mientras que UPPER-BOUND es un límite del sistema y no tiene por que ser el mayor posible. HIGH-VALUE y LOW-VALUE podrían tener los enteros equivalentes 256, o; mientras que UPPER-BOUND y LOWER-BOUND podrían ser 25 = ½ y 86 = .

QUOTE tiene el valor de comillas en tiempo objeto (cuando se ejecuta el programa, por ejemplo en impresión).

ALL no es una constante aunque siempre se la pone entre las constantes figurativas, su formato es:

ALL literal, y quiere decir una secuencia en ese literal.

Ejemplo:

```
ALL   'A B C'
```

podría representar en una orden

```
MOVE 'A B C' TO CAMPO
 A B C A B C A
```

suponiendo que CAMPO tenga siete posiciones.

CUADRO II-a

PALABRAS RESERVADAS COBOL

ACCEPT
ACCESS
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
BEFORE
BEGINNING
BLANCK
BLOCK
BY
CF
CH
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMP
COMPUTATIONAL
COMPUTE
CONFIGURATION
CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
CURRENCY
DATA
DATE-COMPILED
DATE-WRITTEN
DE
DECIMAL-POINT
DECLARATIVES
DEPENDING
DESCENDING
DETAIL
DISPLAY
DIVIDE
DIVISION
DOWN
ELSE
END
ENDING
ENTER
ENVIRONMENT
EQUAL
ERROR
EVERY
EXAMINE
EXIT
FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIRST
FOOTING
FOR
FROM
GENERATE
GIVING
GO
GREATER
GROUP
HEADING
HIGH-VALUE
HIGH-VALUES
HOLD
I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
IS
JUST
JUSTIFIED
KEY
KEYS
LABEL
LAST
LEADING
LEFT
LESS
LIMIT
LIMITS
LINE

LINE-COUNTER
LINES
LOCK
LOW-VALUE
LOW-VALUES
MEMORY
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY
NEGATIVE
NEXT
NO
NOT
NOTE
NUMBER
NUMERIC
OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
OUTPUT
PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POSITION
POSITIVE
PROCEDURE
PROCEED
PROCESS
PROCESSING
PROGRAM-ID
QUOTE
QUOTES
RANDOM
RD
READ
RECORD
RECORDS
REDEFINES
REEL
RELEASE
REMARKS
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
RF
RH
RIGHT
ROUNDED
RUN
SA
SAME
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT-LIMIT
SELECT
SENTENCE
SEQUENTIAL
SET
SING
SIZE
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STATUS
STOP
SUBTRACT
SUM
SYNC
SYNCHRONIZED
TALLY
TALLYING
TAPE
TERMINATE
THAN
THROUGH } equivalent
THRU }
TIMES
TO
TYPE
UNIT
UNTIL
UP
UPON
USAGE
USE
USING
VALUE
VALUES
VARYING
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
ZERO
ZEROES
ZEROS

TEMA III

METALENGUAJE PARA DESCRIBIR LA SINTAXIS DEL COBOL

Leyendo un programa COBOL en inglés, especialmentela PROCEDURE DIVISION, un profano puede tener la impresión de que el ordenador entiende textos escritos en inglés. En la práctica esta expresividad del COBOL es una de sus principales cualidades pues permite a expertos en una determinada materia entender programas (que es lo que hacen) que ellos mejor que nadie han de juzgar si cumplen todos los requerimientos de la aplicación. Sin embargo ellos mismos (suponemos que no saben COBOL) serían incapaces de escribir una línea COBOL que admitiese el ordenador.

Esto es debido a que la sintaxis del COBOL es de una extraordinaria rigidez. Para aprender este lenguaje mejor que pensar en que se trata de estudiar una gramática de una lengua viva, es disponerse a aprender las reglas de un juego como el ajedrez.

Las reglas de juego tienen que darse con total precisión, para esto se utiliza otro lenguaje suficientemente preciso con el que se describe la sintaxis. Este lenguaje es en nuestro caso el español junto con unos símbolos que tienen un significado especial que vamos a definir.

1.- Uso de mayúsculas

Se utilizan para mencionar palabras reservadas COBOL, como ya sabemos estas palabras sólo pueden utilizarse con el significado y en los formatos en que su uso está autorizado.

2. - Palabras obligatorias

Para distinguir a las palabras obligatorias de las incluidas sólo con fines de aclarar o hacer más literario el texto en inglés, se subraya las primeras.

3. - Signos de puntuación y caracteres especiales

La coma y el punto y coma que aparecen en los formatos pueden omitirse, la coma puede sustituirse por AND o , AND.

Nota: En algunos ordenadores no existe el punto y la coma ni el AND; en general, cuando un signo especial ocurre en una descripción sintáctica, éste es obligatorio.

4. - Uso de las minúsculas

Se utilizan las minúsculas para indicar términos genéricos que debe de suministrar el programador (nombres y literales).

5. - Referencias

Para completar o aclarar las descripciones de un formato se pueden incluir notas. Otras veces es necesario referirse a un formato descrito en otro lugar, entonces se escribe el nombre cobol en mayúsculas y separado con un guión y en minúsculas el tipo de descripción.

Ejemplo: PICTURE - cláusula

6. - Uso de corchetes

El texto incluido entre corchetes es opcional. Pero téngase en cuenta que esta opcionalidad no siempre es función del capricho del programador sino también de la aplicación en que se usa el formato.

7. - Uso de llaves

Las llaves indican una elección entre dos o más posibilidades escritas en columna.

8. - El uso de puntos

Tres puntos (. . .) indican que la unidad precedente en la regla sintáctica puede repetirse tantas veces como necesite el programador. Las llaves o los corchetes se utilizan aquí para delimitar la unidad compuesta que puede repetirse.

Por último la semántica (la función o utilidad) de la regla se explica utilizando el castellano.

TEMA IV

ESTRUCTURA DE LA DIVISION DE PROCEDIMIENTOS

Podemos describir la estructura de la división de procedimientos de la siguiente forma:

A	B
PROCEDURE DIVISION.	
(declarativas) (1)	
SEG11	SECTION.
P11.	S111, S112, ..., $S11N_{11}$
P12.	S121, S122, ..., $S12N_{12}$
.	
.	
SEG2	SECTION.
P21.	S211, S212, ..., $S21N_{21}$
P22.	S221, S222, ..., $S22N_{22}$
.	
.	
SEGI	SECTION.
PI1.	SI11, SI12, ..., $SI1N_{I1}$
PI2.	SI21, SI22, ..., $SI2N_{I2}$
.	
.	
PIK_I	SIK_I1, SIK_I2, ..., $SIK_IN_{IK_I}$

Con la notación empleada SEG1, SEG2, . . . , SEGI indican nombres de secciones. Recordemos que los nombres de secciones son nombres de procedimiento que tienen que cumplir las reglas de los nombres, pero en este caso pueden estar formados sólo por números.

Ejemplos:

PREPARACION-DATOS, CALCULO, FINAL-5, 127

El nombre de procedimiento se sigue de la palabra SECTION y se termina con un punto. En esta línea no puede escribirse nada más. El nombre tiene que empezar en el margen A.

P11, P12, . . . PIK_I indican nombres de párrafo. Estos son también nombres de procedimiento y han de cumplir las reglas de estos nombres.

Ejemplos:

CALCULO, FINAL, ERROR-7, 845, 363

Los nombres de párrafo se escriben a partir del margen A y se terminan en un punto. Dejando un espacio al menos, después del punto, puede escribirse sentencias en la misma línea del nombre de párrafo.

S111, S112, . . . , $SIK_I N_{KI}$ indican sentencias. Por ahora podemos entender que una sentencia es una declaración escrita en COBOL que diría algo así: SUME GASTOS A IMPORTE, ESCRIBA RESULTADO.. El primer punto que hay detrás de resultado pertenece a la sentencia y es lo que la define. De no tener punto es sólo una declaración.

En la notación empleada para describir el formato no se han puesto puntos detrás de las sentencias porque este punto las pertenece, esto incluido en una setencia como la S112 que podría decir:

ADD GASTOS TO IMPORTE GIVING RESULTADO.

que significaría: sumar IMPORTE GASTOS dando a RESULTADO el valor de la suma.

IMPORTE, GASTOS y RESULTADO son nombres de dato que el programador define en la DATA DIVISION, en cambio ADD, TO y GIVING son palabras reservadas COBOL y tanto estas como la sintaxis de la sentencia están impuestas por el formato de las declaraciones del verbo ADD (regla de juego) que estudiaremos con otros verbos más adelante.

1.- Secciones

No es necesario dividir siempre la división de procedimientos en secciones, y cuando sólo hay una sección tampoco es necesario poner el nombre de sección. Pero en caso de dividir el programa en secciones, es obligatorio preceder a cada una de estas secciones con un nombre de sección.

Una sección termina cuando empieza otra sección o termina el programa.

NOTA.- El final del programa (distíngase del final de una ejecución en tiempo objeto) en alguna instalación se indica con una sentencia END PROGRAM. que se escribe en el margen A. Pero más general es que se indique con una ficha de control del sistema operativo que avisa al programa compilador del final del programa fuente.

2.- Párrafos

Un párrafo es precedido de un nombre de procedimientos seguido de un punto y un espacio por lo menos después del punto. Después se escriben las sentencias que describen el procedimiento que recibe este nombre de párrafo. El final de un párrafo queda determinado por el principio de otro párrafo, principio de otra sección o por el final de programa.

3.- Sentencia

Las sentencias se escriben a partir del margen B. El orden de ejecución de las sentencias es de izquierda a derecha y de arriba abajo salvo que alguna de las declaraciones contenidas por las sentencias sea de transferencia de secuencia. La dirección de una orden de transferencia es siempre un nombre de procedimientos. De no existir un salto de esta clase al final de la ejecución de la última sentencia de un párrafo pasa a ejecutarse la primera sentencia del párrafo que sigue (hacia abajo).

4.- Declaraciones imperativas simples

Son órdenes que dan lugar a la ejecución de una sola operación de las definidas por los verbos COBOL.

Ejemplos:

```
ADD GASTOS TO IMPORTE,
MOVE IMPORTE TO RESULTADO,
WRITE RESULTADO.
```

que indican la suma de GASTOS con IMPORTE quedando el resultado en importe (1ª declaración), trasladar el contenido de IMPORTE a RESULTADO (2ª declaración), y escribir el resultado (o grabarlo en banda magnética (3ª declaración).

GASTOS, IMPORTE y RESULTADO son nombres de dato dados por el programador en la DATA DIVISION; ADD, TO, MOVE y WRITE son palabras reservadas COBOL.

5. - Serie de declaraciones

Las tres declaraciones del ejemplo anterior, colocadas una seguida de otra (las declaraciones también se ejecutan de izquierda a derecha y de arriba abajo) dan lugar a una serie de declaraciones. Las series pueden ser de la longitud (número de declaraciones) que se necesite.

6. - Declaraciones condicionales

Son declaraciones condicionales las que en tiempo objeto (cuando se ejecuta el programa) pueden dar lugar a distintas operaciones.

Ejemplos: IF A < B ADD A TO C ELSE WRITE RESULTADO que dará por resultado o que A se sume a C o que se escriba RESULTADO seguidamente según cual sea el valor de A respecto de B.

Una declaración condicional siempre tiene incluida otra imperativa por lo menos y el procedimiento completo da lugar a una sentencia que puede contener una o más declaraciones condicionales.

Ejemplos:

```
IF A < B ADD A TO C ELSE IF A = B ADD A TO D.
```

En conclusión, es en la PROCEDURE DIVISION, organizada en la forma descrita, donde el programador describe todo el flujo de

operaciones que constituye la ejecución del programa en cada caso que pueda presentarse.

(1) Las declarativas son de uso optativo y no van a ser tratadas en esta introducción. Cuando se usan se agrupan al principio de la PROCEDURE DIVISION y se organizan a su vez en forma de secciones. Van precedidos por la palabra clave DECLARATIVES y van seguidas por las palabras clave END DECLARATIVES.

Ejemplo de aplicación:

Se trata de una librería que mantiene un archivo de existencias en banda magnética. En este archivo se guarda la siguiente información:

1. - Número clave del libro. 2. - Título. 3. - Autor. 4. - Materia. 5. - Editorial. 6. - Nacionalidad. 7. - Precio compra. 8. - Precio fuente. 9. - Número de ejemplares.

Cada libro existente en la librería tiene una tarjeta con su número clave perforado en las seis primeras columnas. Cuando se efectúa una venta se saca esta tarjeta y se almacena con las de otros libros también vendidos en el día. Al final de la jornada se ordena este paquete de tarjetas (como simplificación suponemos que sólo se venden un ejemplar de cada libro) y se procesa con un ordenador obteniendo los siguientes resultados:

a. - El archivo anterior queda actualizado en uno nuevo.
b. - Se obtiene un listado como sigue:

LIBRERIA FUENTES

VENTAS

NUMERO CLAVE	TITULO	AUTOR	MATERIA	EDITORIAL	PRECIO FUENTE
A-068	LOSANA	KEY	NOVELA	PRAO	865,60
A-074	STES	CLAY	MATH.	LONG	1380,40
A-081	AVES	AGRO	ZOO	RIEL	860,00
A-183	LA LEY	JUEZ	DERE.	ARRAIZ	900,00
				TOTAL	4006,00

Al archivo en banda anterior le llamamos ARCHIVO-GENERAL-V y al nuevo ARCHIVO-GENERAL-N y a sus únicos registros LIBRO y LIBRO-N. Al archivo de fichas de dato le llamamos MOVIMIENTO y a su registro LIBRO-ACTUALIZADO. Al informe que sale por la impresora le llamamos VENTAS y a su registro LINEA.

El programa que nos ejecutaría esta operación podría estar escrito así en la PROCEDURE

```
PROCEDURE DIVISION.
EMPIEZA.
      OPEN INPUT ARCHIVO-GENERAL-V, MOVIMIENTO
      OUTPUT ARCHIVO-GENERAL-N, VENTAS.
      MOVE ZEROS TO TOTAL
      MOVE SPACES TO LINEA
      WRITE LINEA AFTER SALTO.
LEER-FICHA.
      READ MOVIMIENTO RECORD AT END MOVE TOTAL TO TOTAL-
      EDIT WRITE LINEA
      FROM L-TOTAL AFTER ADVANCING 2 LINES,
      GO TO TERMINAR-CINTA.
LEER-CINTA.
      READ ARCHIVO-GENERAL-V AT END GO TO ERROR.
      IF NUMERO-CLAVE OF LIBRO-ACTUALIZADO
      IS LESS THAN NUMERO-CLAVE OF LIBRO
      GO TO ERROR.
      IF NUMERO-CLAVE OF LIBRO-ACTUALIZADO
      IS EQUAL TO NUMERO-CLAVE OF LIBRO
      MOVE CORR LIBRO TO LIBRO-VENDIDO
      WRITE LINEA FROM LIBRO-VENDIDO AFTER 1,
      ADD PRECIO-FUERTE OF LIBRO TO TOTAL
      SUBTRACT 1 FROM NUM-EJEMPLARES OF LIBRO
      WRITE LIBRO-N FROM LIBRO
      GO TO LEER FICHA 9
      WRITE LIBRO-N FROM LIBRO
      GO TO LEER CINTA.
TERMINAR-CINTA.
      READ ARCHIVO-GENERAL-V; AT END GO TO CERRAR.
      WRITE LIBRO-N FROM LIBRO
      GO TO TERMINAR-CINTA.
ERROR.
      DISPLAY 'ERROR EN DATOS' UPON CONSOLE.
```

```
CERRAR.
        CLOSE ARCHIVO-GENERAL-V, ARCHIVO-GENERAL-N,
        MOVIMIENTO, VENTAS
        STOP RUN.
END     PROGRAM.
```

El alumno a estas alturas debe contentarse con examinar esta PROCEDURE buscando párrafos, sentencias, declaraciones imperativas, declaraciones condicionales y series de declaraciones. Pero a medida que vaya avanzando en el curso debe ir descifrando la lógica del proceso lo que no debe ofrecerle mayores dificultades.

TEMA V

VERBOS Y OPERADORES ARITMETICOS

En COBOL hay dos formas de programar los cálculos aritméticos: usando verbos aritméticos o usando el verbo COMPUTE. Estas dos formas se pueden emplear alternativa o combinadamente. La elección no es siempre fruto del capricho del programador, a veces el uso de COMPUTE se hace indispensable, pero cuando esto no es así el texto suele resultar más literario y claro usando los verbos aritméticos.

Las formas que vamos a ofrecer al alumno son simplificaciones de las muchas opciones que ofrece el COBOL-65.

1.- ADD

1) ADD { literal-1 / nombre-dato-1 } [, { literal-2 / nombre-dato-2 }] ... TO nombre-dato-m [, nombre-dato-n] ...

2) ADD { literal-1 / nombre-dato-1 } { literal-2 / nombre-dato-2 } [, { literal-3 / nombre-dato-3 }] ... GIVING nombre-dato-m

Ejemplos:

ADD 126.0, PRECIO, COSTE TO IMPORTE, BASE

(sumar 126, 0, contenido de PRECIO, más contenido de COSTE a IMPORTE y dejarlo en IMPORTE y a BASE dejándolo en BASE.
Sea PRECIO= 200.6 COSTE= 140. IMPORTE= 400 y BASE = 200, después de la operación IMPORTE= 866, 60 y BASE = 666, 60).
NOTA.- En los ejemplos las variables que no se mencionan después de la operación conservan su valor primitivo.

ADD PRECIO, 126, 0, COSTE GIVING BASE

(con los valores anteriores, después de la operación BASE = 466, 60).

2.- SUBTRACT

1) SUBTRACT { literal-1 / nombre-dato-1 } [, { literal-2 / nombre-dato-2 }] .... FROM

nombre-dato-m [, nombre-dato-n] ....

2) SUBTRACT { literal-1 / nombre-dato-1 } [, { literal-2 / nombre-dato-2 }] .... FROM

{ literal-m / nombre-dato-m } GIVING nombre-dato-n

Ejemplos:

SUBTRACT HORAS-25, HORAS-40 FROM TOTAL, INTEGRO

(Si HORAS-25 = 250.0; HORAS-40 = 475.0; TOTAL 1200.0
INTEGRO = 975.0 después de la operación TOTAL = 475.0
INTEGRO = 250)
SUBTRACT SEGUROS, UTILIDADES FROM SUELDO GIVING LIQUIDO
(si SEGUROS = 200; UTILIDADES = 400; SUELDO = 1000. y
LIQUIDO = 400 después de la operación LIQUIDO = 400.)

3.- MULTIPLY

1) MULTIPLY { nombre-dato-1 / literal-1 } BY nombre-dato-2 [, nombre-dato-3]

2) MULTIPLY { nombre-dato-1 / literal-1 } BY { nombre-dato-2 / literal-2 }

GIVING nombre-dato-3

Ejemplos:

MULTIPLY 30.0 BY CANTIDAD

(Si CANTIDAD = 20 después de la operación CANTIDAD = 600.)

MULTIPLY PRECIO BY CANTIDAD

(Si PRECIO = 30 y CANTIDAD = 20 después de la operación CANTIDAD = 60)

MULTIPLY PRECIO BY CANTIDAD GIVING IMPORTE

(Si PRECIO = 30, CANTIDAD = 20 e IMPORTE = 400 después de la operación IMPORTE = 600)

4.- DIVIDE

1) DIVIDE { nombre-dato-1 / literal-1 } INTO nombre-dato-2 [, nombre-dato-3]

2) DIVIDE { nombre-dato-1 / literal-1 } INTO { nombre-dato-2 / literal-2 }

GIVING nombre-dato-3

3) DIVIDE nombre-dato-1 BY { nombre-dato2 / literal-1 }

4) DIVIDE { nombre-dato-1 / literal-1 } BY { nombre-dato-2 / literal-2 }

GIVING nombre-dato-3

Ejemplos:

DIVIDE 5 INTO TOTAL , SUMA-B, SUMA-C

(Si TOTAL = 200, SUMA-B = 40, SUMA-C = 150 después de la operación TOTAL = 40 , SUMA-B = 8 y SUMA-C = 30)

DIVIDE PARTES INTO TOTAL GIVING DIVISION

(Si PARTES = 12, TOTAL = 360 y DIVISION = 720 después de la operación DIVISION = 30)

DIVIDE COSTE BY UNIDADES

(Si COSTE = 2000 y UNIDADES = 50 después de la operación COSTE = 40)

DIVIDE COSTE BY UNIDADES GIVING PARTIDA

(Si COSTE = 2000, UNIDADES = 50 y PARTIDA = 80 después de la operación PARTIDA = 40)

5. - Expresiones aritméticas

Se llaman expresiones aritméticas a nombres de datos, literales numéricos o series de literales y datos numéricos separados por operadores aritméticos que pueden ser reducidos a un valor numérico único.

Los formatos posibles pueden definirse en COBOL así:

1) nombre-dato

2) literal

3) {nombre-dato / literal} operador {nombre-dato / literal}

4) {nombre-dato / literal} operador {nombre-dato / literal} operador .....

..... operador {nombre-dato / literal}

Los operadores aritméticos son +, -, *, /, **; recuérdese que se escriben dejando un blanco a cada lado.

Las operaciones se realizan con el siguiente orden de precedencia:

(1) exponenciación.
(2) multiplicación y división.
(3) suma y resta.

Dentro del mismo orden las operaciones se realizan de izquierda a derecha.

Si A = 4, B = 6, C = 2, D = 3 y E = 12

A + B * C ** D resulta igual a 52
A + B * C " " " 16
E - A / C " " " 10

Hay operaciones como $\frac{A + B}{A - B}$ que no pueden escribirse utilizando sólamente los nombres y operadores. En estos casos se utilizan los paréntesis. También se utilizan para alterar el orden normal de ejecución de operaciones.

Se define que los paréntesis tienen el mayor orden de precedencia. Por ejemplo la operación anterior se podría escribir ahora:

(A + B) / (A - B)

Si A = 4 , B = 3 y C = 6

((A + B) * A - C) / 2 da como resultado 11.

6.- COMPUTE

Un formato simplificado es:

COMPUTE nombre-dato-1 = { literal / nombre-dato-2 / expresión aritmética

Obsérvese el parecido que tiene con las sentencias FORTRAN, y el significado es idéntico. El signo = no significa igual si no que es un signo de afectación: el nombre de datos a la izquierda toma el valor único que puede tomar la expresión a la derecha.

Ejemplo: COMPUTE H = A + B + C ** D

(Si A = 4, B = 6, C = 2, D = 3 y H = 12 después de la operación H = 18)

TEMA VI

VERBOS DE CONTROL DE SECUENCIA

Hemos dicho antes que la secuencia normal de operaciones se efectúa declaración tras declaración, de iaquierda a derecha y de arriba abajo. Sin embargo hay declaraciones especiales que alteran esta secuencia. Estas declaraciones se forman con los verbos: GO TO, PERFORM y STOP. Dan también lugar a alteración normal de la secuencia las declaraciones condicionales que se forman con la palabra clave IF, por lo que también las estudiaremos en este tema.

1. - GO TO

Tiene dos opciones:

a) GO TO [nombre-procedimiento]

b) GO TO nombre-procedimiento-1 [nombre-procedimiento-2] . [nombre-procedimiento-n] DEPENDING ON nombre-dato

Nombre-procedimiento debe ser un párrafo y el resultado es que la siguiente declaración que se ejecuta es la primera de este párrafo continuando con la secuencia normal desde allí hasta que sea alterada por otra declaración de salto.

En la opción b) nombre-dato debe ser entero positivo que será definido antes de ejecutarse esta declaración. Este número indica el número de orden que ocupa el nombre de párrafo a donde salta la secuencia. Obsérvese la semejanza que tiene con el GO TO condicional de FORTRAN.

Ejemplos:

GO TO CALCULO

La siguiente declaración que se ejecuta es la primera declaración del párrafo CALCULO

GO TO RUTINA-1, RUTINA-2, RUTINA-3, DEPENDING ON K

Si K en el momento de la ejecución vale 2 la secuencia salta a la primera declaración del párrafo RUTINA-2. Si el valor fuese distinto de 1, 2, ó 3 no se ejecuta ningún salto y el programa continúa en la siguiente declaración al GO TO.

2.- IF

Para poder tratar a las declaraciones condicionales con toda generalidad, vamos a referirnos a la IF-sentencia que vamos a definir. Téngase en cuenta que esta generalidad sólo la proporciona algunos compiladores y que como frecuentemente advertimos será necesario para el programador la consulta del manual de COBOL que va a utilizar para ver si estas posibilidades están permitidas. La avanzada generalidad que presentamos está justificada por la homogénea presentación que permite.

IF-sentencia.-

IF expresión-condicional (THEN) { declaración-1 (declaración-2..) / NEXT SENTENCE }

[{ ELSE / OTHERWISE } { Declaración-3 (declaración-4 ...) / NEXT SENTENCE }]

Las declaraciones pueden ser a su vez condicionales con el mismo formato que la IF-sentencia pero sin el punto de sentencia.

Ejemplos:

IF A < B GO TO SUMAR.
WRITE LINEA FROM SUELDO AFTER 1.

Si A es menor que B hay salto de secuencia para ejecutar la primera declaración del párrafo SUMAR y continúa allí.

Si no es menor que B se ejecuta la sentencia siguiente: se escribe LINEA ....

```
IF A<B  GO TO SUMAR ELSE ADD AUMENTO TO SUELDO.
WRITE LINEA FROM SUELDO AFTER 1.
```

Si A es menor que B se salta el párrafo SUMAR. Si no es menor se ejecuta la suma de AUMENTO y SUELDO, después la siguiente sentencia: escribe LINEA....

```
IF A<B  ADD AUMENTO TO SUELDO ELSE GO TO SUMAR.
WRITE LINEA FROM SUELDO AFTER 1.
```

Si A es menor que B se suma AUMENTO con SUELDO después se ejecuta la siguiente sentencia: escribe LINEA....

En caso de que no sea A<B entonces se pasa a ejecutar la primera declaración del párrafo SUMAR.

```
IF A<B THEN IF A<C NEXT SENTENCE ELSE GO TO SUMAR, ELSE IF A<C NEXT SENTENCE ELSE ADD AUMENTO TO SUELDO GO TO SUMAR. WRITE LINEA FROM SUELDO AFTER 1.
```

Si A<B y A<C se ejecuta la siguiente declaración: escribe....
Si A<B y A=C ó A>C se salta párrafo SUMAR
Si A$\not<$B y A<C se ejecuta la siguiente sentencia: escribe ....
Si A$\not<$B y A$\not<$C se suma AUMENTO con SUELDO y se salta a SUMAR

Ahora será conveniente parar para definir lo que llamamos expresión condicional.

Una expresión condicional es:

a) un nombre de condición
b) una condición de clase
c) una condición de signo
d) una condición de relación
e) una serie de expresiones condicionales asociadas con las conectivas lógicas AND y OR.

La propiedad común que tienen las expresiones condicionales es que se reducen a un solo valor: verdadero o falso. Por esta cualidad similar al de las aritméticas que también toman un valor único, es por lo que se llaman expresiones.

a) nombres de condición.

Hay de dos clases: definidos en la DATA DIVISION y los definidos en la ENVIRONMENT DIVISION. Ambas serán definidas en estas divisiones, ahora indicaremos sólo que los definidos en la DATA son nombres que damos a especiales valores que puede tomar un nombre-dato. Por ejemplo sea el nombre-dato que contenga claves de nacionalidad, le podíamos haber dado el nombre de NACIONALIDAD y por ejemplo cuando vale E indica que es español. Luego veremos que a este valor le podemos asociar un nombre (nombre-condición) como ESPAÑOL que será verdadero cuando NACIONALIDAD tenga el valor E, en otro caso es falso.

En el párrafo SPECIAL-NAMES de la ENVIRONMENT podemos asociar un nombre-condición a la posición de un SWITCH o a una situación especial como el haber detectado el final de página. Este último caso sería normal darle el nombre de FIN-PAGINA. En otros casos importantes este particular se define en el párrafo I-O-CONTROL asociándolo a FORM-OVERFLOW. El nombre-condición definido en cada momento tendrá un solo valor.

b) condición de clase.

El formato de estas expresiones es:

nombre-dato IS(NOT) { NUMERIC / ALPHABETIC }

que pregunta sobre si el contenido es exclusivamente numérico (0 al 9) o alfabético (␣ y de la A a la Z).

c) condición de signo.

El formato de estas expresiones es:

{ nombre-dato / expresión aritmética } IS (NOT) { POSITIVE / NEGATIVE / ZERO }

Ejemplos:

A - B IS NEGATIVE

Si A y B positivos esta expresión será verdad cuando B>A

d) condición de relación.

Tienen el siguiente formato:

{nombre-dato-1 / expresión-aritmérica-1 / constante-figurativa-1 / literal-1} IS (NOT) relación {nombre-dato-2 / expresión aritmética / constante figurativa / literal-2}

Las relaciones usadas en COBOL son:

GREATER THAN
EQUAL TO
LESS THAN

cuyos equivalentes respectivos son >, =, < y; unos u otros se usan indistintamente.

Ejemplos: PRECIO GREATER THAN TOPE
DESCUENTO EQUAL TO ZERO
AUMENTO IS NOT LESS THAN LIMITE

e) Estas expresiones llamadas compuestas, no serán estudiadas en esta introducción.

Para dar un ejemplo todavía más general de la IF-sentencia vamos a utilizar unos símbolos con significado especial. C_i, indica una expresión condicional. S_i, indica una declaración. Vamos a analizar ahora la IF-sentencia

IF1 C1 S1 IF2 C2 IF3 C3 S2 ELSE IF4 C4 S3 ELSE S4 ELSE
IF5 C5 S5 ELSE S6

Lo primero que debemos hacer es emparejar cada ELSE con un IF. Puede haber un IF sin ELSE pero el revés indicaría un error de sintaxis (estúdiese el formato de la IF-sentencia). Cuando delante de un ELSE encontramos otro ELSE en vez de un IF entonces tratamos de emparejar este ELSE hasta que nos encontremos con un IF sin pareja; entonces volveremos en busca del primer ELSE sin pajera y volvemos a avanzar hasta que todos los ELSE quedan emparejados. ¿Qué ocurre cuando quedan IF sin pareja? Pues que cuando el valor de la expresión condicional que les corresponda sea falso la siguiente declaración que se

ejecuta será la primera de la siguiente sentencia. Con este esquema podemos construir fácilmente el organigrama que nos describe el flujo de ejecución de esta sentencia (figura VI-1).

La aparente complejidad que tenía la sentencia (mucho mayor que las que en la práctica se presentan usualmente) convencerá plenamente de que la IF-sentencia no ofrece dificultades haciéndola un análisis ordenado.

3. - PERFORM

Es éste el verbo más importante COBOL. Tiene algunas semejanzas con el DO y las subrutinas FORTRAN pero implica un uso combinado de ambas.

Opción 1.

PERFORM nombre-procedimiento-1 (THRU nombre-procedimiento-2)

Opción 2.

PERFORM nombre-procedimiento-1 (THRU nombre-procedimiento-2)

{entero / nombre-dato} TIMES

Opción 3.

PERFORM nombre-procedimiento-1 (THRU nombre-procedimiento-2)
UNTIL expresión condicional

Opción 4.

PERFORM nombre-procedimiento-1 (THRU nombre-procedimiento-2)
VARYING nombre-dato-1 FROM {literal-1 / nombre-dato-2}
BY {literal-2 / nombre-dato-3} UNTIL expresión condicional

Opción 5.

PERFORM nombre-procedimiento-1 (THRU nombre-procedimiento-2)

VARYING nombre-dato1 FROM {literal-1 / nombre-dato-2}

BY {literal-2 / nombre-dato-3} UNTIL expresión condicional

[AFTER nombre-dato-4 FROM {literal-3 / nombre-dato-5} BY {literal-4 / nombre-dato-6}

UNTIL expresión condicional

[AFTER nombre-dato-7 FROM {literal-5 / nombre-dato-8} BY {literal-6 / nombre-dato-9}

UNTIL expresión condicional.

En los formatos anteriores nombre de procedimiento es un nombre de párrafo o un nombre de sección.

Literal: indica a literales numéricos solamente, cuando los valores tienen que ser enteros positivos se ponen enteros.

Nombre-dato: en la opción 2 tiene que tomar valores enteros positivos.

Nombre-dato: en las otras opciones puede tomar cualquier valor numérico.

Opción 1. Para aclarar el concepto de PERFORM vamos a considerar un supuesto. En un programa, en distintos lugares de la PROCEDURE tenemos que efectuar el siguiente cálculo:

```
CALCULO.
        COMPUTE A = B + (C + D) * (D + 1 + E) * 5
        COMPUTE R = (A -1200) * H
        IF R < 16.0 NEXT SENTENCE ELSE GO TO
        CALCULO.
```

Es evidente que la labor de escribir estas sentencias en cada lugar necesario sería rutinaria. Sin embargo siendo como es una subrutina sería mucho más conveniente poderla dar un nombre y dar orden de ejecutarla mencionando este nombre. En el caso simple del ejemplo, el nombre que se da a la rutina es el párrafo y la orden de ejecución se da con: PERFORM CALCULO.

El párrafo CALCULO puede encontrarse en cualquier lugar de la PROCEDURE y sus límites quedan determinados por las reglas que determinan el final de los párrafos.

Si en vez de ejecutar un sólo párrafo queremos ejecutar una serie consecutiva de los mismos utilizaríamos la opción THRU. Por ejemplo sean los párrafos supersimplificados:

```
P1.   ADD A TO B.
P2.   WRITE REGISTRO.
P3.   SUBTRACT 50.0 FROM A.
P4.   MULTIPLY 17 BY RESULTADO.
P5.   GO TO P2.
P6.   COMPUTE R = (12 + E) ** 17
```

Si en un lugar del programa quisiéramos ejecutar P1, P2, P3, P4 podríamos escribir la declaración:

PERFORM P1 THRU P4

Entonces se ejecutaría desde la primera declaración de P1 hasta la última de P4.

Después de haberse cumplido un PERFORM se ejecuta la instrucción que sigue a este PERFORM

Ejemplos:

```
PERFORM P1 THRU P2
PERFORM P6
ADD A TO B
```

Hará que se ejecuten los párrafos P1, P2, P6 y por último se sumará A a B.

En cambio, si decimos PERFORM P3 THRU P5, ADD A TO B; el control no vuelve a la siguiente declaración al PERFORM porque el

GO TO de P5 lo impide. Es importante atender a este particular que aun que este caso se remedie poniendo la regla de que la última declaración no sea una transferencia de secuencia, es mejor que el alumno observe que en estos casos hay siempre un contrasentido entre los aparentes objetivos del programa. Si decimos que se ejecuten los párrafos P3 a P5 con un PERFORM, estamos deciendo que además después el control pase a la declaración siguiente al PERFORM, por lo que no se nos ocurrirá escribir en P5 como última declaración GO TO P2, que está en contradicción con lo que queremos.

Sin embargo utilizando declaraciones condicionales se pueden presentar problemas que vamos a ver con un ejemplo:

```
DESCUENTO  UTILIDADES
           IF SUELDO-MENSUAL GREATER THAN 7500
           SUBTRACT 7500 FROM SUELDO-MENSUAL
           GIVING LIQUIDO
           COMPUTE DESCUENTO = (14,0 * LIQUIDO) / 100.0
           GO TO SALIDA
           ELSE
           IF SUELDO-ACUMULADO GREATER THAN DEDUCCION
           COMPUTE DESCUENTO = (14.0 * SUELDO-MENSUAL) / 100.0
           GO TO SALIDA.
           MOVE ZEROS TO DESCUENTO.
SALIDA.EXIT.
```

Obsérvese que el párrafo SALIDA es imprescindible porque a la IF-sentencia la sigue MOVE ZEROS TO DESCUENTO y esto no debe ejecutarse en los otros casos, por lo tanto hay que saltarla con un GO TO.

El párrafo SALIDA por otra parte no tiene otro fin que devolver la secuencia a la declaración siguiente al PERFORM que llame a este subrutina. Esto se logra utilizando el verbo EXIT. Este verbo indica sólo el punto final del procedimiento.

Su formato es EXIT y debe ser la única palabra en el párrafo.

Qué hubiera ocurrido si en la secuencia P1, P2 ... P6 del ejemplo anterior hubiéramos entrado siguiendo la secuencia normal del programa o simplemente después de un GO TO P1?

Pues se ejecutará P1, después P2, P3, P4, P5, P2, P3... y habríamos entrado en un ciclo. Con esto queremos aclarar que el retorno que se tiene a la declaración siguiente al PERFORM es temporal y sólo

mientras se está ejecutando este PERFORM. Así podemos escribir: PERFORM P1 THRU P3 y el retorno está después de la última declaración de P3, seguidamente la siguiente declaración podría ser: PERFORM P1 THRU P4, y ahora el retorno estaría después de la última declaración de P4.

Con el EXIT-párrafo ocurre algo semejante. Si en la rutina siguiente entramos por un GO TO o una secuencia normal, no un PERFORM, el párrafo SALIDA resulta en una operación nula y se ejecuta el siguiente párrafo, el P3.

```
P1.     ADD A TO B.
P2.     DIVIDE 4 INTO H.
SALIDA. EXIT.
P3.     DIVIDE H BY 4.
```

Las consideraciones anteriores llevan al establecimiento de las reglas de anidado. Se dice que las declaraciones PERFORM están anidadas cuando la secuencia de sentencias de un PERFORM está dentro de la secuencia de sentencias de otro PERFORM previamente dado.

Con la siguiente descripción quedará claro este concepto.

Anidado correcto

```
26.
27.  PERFORM 71 THRU 78.
28.
29.
30.
31.
32.
.
.
71.
.
.
74.  PERFORM 29 THRU 31.
.
78.
.
87.  PERFORM 26. THRU 32.
.
```

Anidado incorrecto

```
26.
27.  PERFORM 29 THRU 31.
28.
29.
30.
31.
32.
.
.
.
64.  PERFORM 26 THRU 30.
.
.
.
70.
71.  PERFORM 72 THRU 75.
72.
73.
```

Anidado correcto	Anidado incorrecto
.	74.
102. PERFORM 119 THRU 121.	75.
.	.
.	.
119.	.
120.	92. PERFORM 70 THRU 75.
121.	.
122.	.
123.	.
.	110. PERFORM 72 THRU 75.
.	
152. PERFORM 120 THRU 123.	

Todo cuanto hemos dicho aquí para la opción 1 es igualmente válido para las opciones que comentamos a continuación.

Opción 2. - Es igual que la opción 1 excepto que el párrafo o secuencia de párrafos se ejecuta tantas veces seguidas como indica el entero o el nombre-dato que ha de contener también un entero.

Opción 3. - Es igual que la anterior excepto que el REFORM se ejecuta mientras la expresión condicional tenga un valor falso, deja de efectuarse en cuanto toma un valor verdadero.

Ejemplos:

```
A    B

          MOVE 0.5 TO A, MOVE 0.2 TO B
          PERFORM CALCULO UNTIL PROBA + PROBB > 1.0
CALCULO.
          COMPUTE PROBABILIDAD = PROBA + PROBB -
          PROBA * PROBB
          COMPUTE PROACUM = PROACUM + PROBABILIDAD
          WRITE LINEA FROM RESULTADO AFTER 1
          ADD . 01 TO PROBA, PROBB.
SIGUE.
```

Si cálculo anterior se efectuará escribiéndose los resultados para los valores

PROBA	PROBB
0.50	0.20
0.51	0.21
.	.
.	.
0.65	0.35

Opción 4. - Esta opción es una variante de la opción 3 en la que se añade un nombre de datos que va tomando valores en cada ejecución desde el indicado por el literal-1 o nombre-dato-3.

Ejemplos:

```
        REFORM CALCULO VARYING I FROM o BY-2
        UNTIL A <o.
CALCULO.
        COMPUTE A = 3 * I + 600.
```

Los valores que tomarán I y A serán los siguientes:

I	A
0	600
-2	594
-4	588
.	.
.	.
-198	6
-200	0

Opción 5. - La opción 5 es como la 4 pero ahora son hasta tres los nombres de datos que pueden variar. Estos toman valores por el orden en que aparece, por lo tanto, cada vez que nombre-dato-4 cambia de valor el nombre-dato-7 efectúa un ciclo. Cada ciclo de nombre-dato-4 da origen a un cambio en el nombre-dato-1.

Decimos un cambio en el nombre-dato para fijar la atención del alumno en que los incrementos pueden ser positivos o negativos.

Ejemplo:

```
        MOVE 10 TO BASE
        MOVE 2 TO INCRE
        PERFORM CALCULO VARYING I FROM 1 BY 1
        UNTIL I > 17
        AFTER J FROM -3 BY INCRE UNTIL J = 50
        AFTER K FROM BASE BY -1.0 UNTIL K <- 30
        AFTER
.
.
.
CAL     CULO.
        COMPUTE A = 2 * I + J - K.
SIG     UE.
```

Los valores que toman I, J, K y A son los siguientes:

I	J	K	A
1	-3	10	-11
1	-3	9	-10
	.	.	.
1	-3	-30	29
1	-1	10	-9
.	.	.	.
.	.	.	.
1	49	-30	81
2	-3	10	-9
.	.	.	.
.	.	.	.
17	49	-30	113

4.- STOP

Formato:

STOP { literal / RUN }

Ejemplos:

SOTP 17

Un 17 se escribe en la máquina de consola o se encienden unas luces (según la instalación) con esta codificación; se para la ejecución y pulsando la tecla START se continúa desde la declaración siguiente a la STOP. STOP-literal funciona como un PAUSE en FORTRAN.

STOP 'COLOCAD CINTA CALCULO-2'

Este mensaje escrito por la máquina de consola informaal operador que esta banda va a actuar seguidamente.

STOP RUN. detiene definitivamente la ejecución del programa porque éste ha llegado a su final lógico. Debe comprenderse que esta sentencia puede estar colocada en cualquier parte del programa fuente, casi nunca al final y puede haber más de un STOP RUN.

Recordemos también que el final del programa fuente se indica con END PROGRAM o una ficha de control del sistema operativo que informa al compilador de que no tiene que traducir más sentencias.

En los grandes sistemas operativos la orden de STOP RUN no para en realidad al ordenador. El compilador traduce este STOP por un salto al programa monitor que toma control y se continúa automáticamente con otros trabajos.

5.- NOTE

Formato:

NOTE cadena de caracteres

Los caracteres pueden ser cualquiera de los usados por COBOL menos el punto.

NOTE se utiliza para comentarios.

```
17.  ADD A TO B
     SUBTRACT 500.0 FROM LIQUIDO.
     NOTE LIQUIDO ES EL UTILIZADO PARA FACTURAR.
     COMPUTE FACTURA = LIQUIDO 0.65 + GASTOS
```

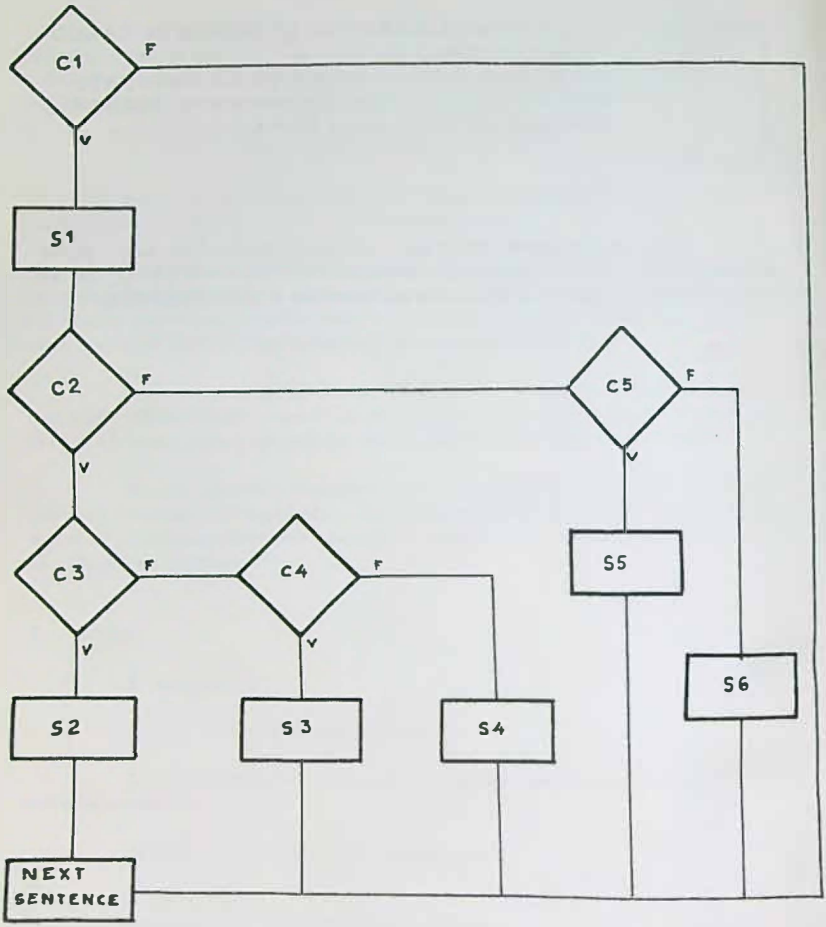
Cuando NOTE es la primera sentencia del párrafo, todo el párrafo es comentario.

```
NOTA - 1.      NOTE CON LA SUBRUTINA QUE SIGUE SE CALCU-
               LAN LOS SUELDOS
               LIQUIDOS DE LOS EMPLEADOS DE LA EMPRESA.
               PARA EL CALCULO DE LOS LIQUIDOS DE PERSONAL
               EVENTUAL SE EMPLEA LA RUTIN CALCULO-2

CALCULO - 1.
```

Los comentarios no dan lugar a ninguna traducción por parte del compilador. Si en el programa ponemos PERFORM NOTA-1, no se ejecuta nada y se pasa a la siguiente declaración a este PERFORM.

FIGURA VI-1



TEMA VII

ESTRUCTURA DE LA DIVISION DE DATOS

En esta instrucción informal hemos procurado desarrollar una comprensión intuitiva por parte del alumno sobre cómo se escribe un programa en lenguaje COBOL. A medida que hemos ido avanzando, la idea de 'regla de juego' se ha ido poniendo en práctica estableciendo así formatos exactos que siguen el uso de verbos como ADD, GO TO, PERFORM, etc. Con los datos que se manejan ocurre otro tanto. Hay que definirlos con toda exactitud indicando un tamaño, clase y uso. Estas definiciones se hacen en la DATA DIVISION, que como la PROCEDURE se divide en secciones pero aquí los nombres de las secciones son palabras fijas reservadas.

1. - Secciones

Las secciones de la DATA DIVISION son:

```
DATA DIVISION.
FILE SECTION.
REPORT SECTION.
WORKING-STORAGE SECTION.
CONSTANT SECTION.
```

La REPORT SECTION no se estudia en esta introducción. La descripción de los archivos con el detalle de los registros que lo forman se hace en la FILE SECTION. Los hombres de datos índices o indepen-

dientes, los campos utilizados temporalmente, o los que se reservan para operaciones de entrada y salida que en la FILE SECTION se destruyen al realizar estas operaciones, se describen en la WORKING-STORAGE SECTION.

Hay cantidades que se usan en el programa que tienen un significado bien determinado como puede ser el tipo de interés, en vez de utilizar, sea 4.25, es más conveniente poner TIPO-INTERES. Lo mismo ocurre con ciertos literales no numéricos. Todos estos nombres de dato con valor constante se definen en la CONSTANT section.

2. - Archivos, registros, grupos y elementos

Los procesos de gestión consisten en su generalidad en una información que se mantiene en un archivo llamado maestro, en una actualización que se da por medio de otro archivo (generalmente en fichas), y en un informe o grupo de informes que se suelen dar por impresora, lo que también constituye un archivo.

Es por lo tanto el archivo la unidad principal y mayor que se trata en un programa de COBOL.

Pero un archivo se compone de registros. El detalle, la característica de éstos, definen a su vez la estructura de estos archivos. Las cualidades de los registros no son independientes del medio en que se almacena un archivo. Por ejemplo, si el archivo es en ficha, los registros como máximo tendrán 80 caracteres, si es un informe impreso, el registro, que es la línea de impresión, solo podrá tener 120, 128, 132, etcétera, caracteres, según sea la capacidad de la impresora.

A su vez los registros tienen sus cualidades dependientes de los elementos que lo componen. Para examinar mejor la composición de un registro estudiaremos el supuesto del archivo de un almacén de materiales especiales. Este almacén mantiene la siguiente información en cada registro del archivo. Todos los registros son iguales.

(1, 6) Clave de la pieza
(7, 16) Fabricante
(17, 22) Fecha fabricación
(23, 25) Ancho
(26, 28) Alto
(29, 31) Largo
(32, 34) Dureza

NOTA. - Los números entre paréntesis indican la primera y última columnas que ocupa la información.

(35, 40) Fecha recepción
(41, 46) Fecha venta
(47, 51) Precio compra
(52, 56) Precio venta
(57, 60) Existencias
(61, 64) Punto pedido

En esta descripción se han nombrado todos los elementos que componen el registro, de esta manera queda este complemento descrito en su composición. Sin embargo lo normal es documentar mejor esta descripción con la siguiente estructura:

```
( 1, 22 )        Identificación
                    ( 1, 6 )  Clave de la pieza
                    ( 7, 16 ) Fabricante
                    ( 17, 22 ) Fecha fabricación
( 23, 34 )       Características técnicas
          ( 23, 31 ) Dimensiones
                    ( 23, 25 ) Ancho
                    ( 26, 28 ) Alto
                    ( 29, 31 ) Largo
          ( 32, 34 ) Dureza
( 35, 64 )       Control almacén
          ( 35, 40 ) Fecha recepción
          ( 41, 46 ) Fecha de venta
          ( 47, 51 ) Precio compra
          ( 52, 56 ) Precio venta
          ( 57, 60 ) Existencias
          ( 61, 64 ) Punto pedido
```

En esta nueva descripción a 'Identificación', 'Características técnicas', 'Dimensiones', y 'Control almacén' se les llama grupos. COBOL permite tratar la información a nivel de registro, de grupo y de elemento. Pero no todas operaciones son posibles en todos los niveles, por ejemplo los verbos aritméticos ADD, SUBTRACT, MULTIPLY y DIVIDE presentados sólo permiten las operaciones a nivel de elemento. Pero además de ser elementos los nombres-dato, tienen que tener otra cualidad (ser numéricos) que en el próximo tema estudiaremos.

3. - Nombres de dato

Como hemos visto a la información se la estructura en archivos, registros, grupos y elementos. Para poder manejar esta información se puede dar nombre a estas unidades, a estos nombres se les llama nombre de datos (nombre-dato) y se construyen aplicando las reglas COBOL para formar palabras. En los nombres-dato uno al menos de los caracteres tiene que ser letra. Los nombres dato los da el programador procurando que sea significativo de la información que ha de contener. Esto en COBOL no es difícil de conseguir dada la longitud que pueden alcanzar las palabras (treinta caracteres). Sin embargo si la longitud de las palabras usadas es muy larga, se hace muy penosa la escritura de la PROCEDURE si estos nombres son utilizados con frecuencia, y el texto pierde también legibilidad.

4. - Niveles

Cuando hemos descrito el registro de las piezas del almacén estructurándole por grupos, lo hemos sangrado para aclarar esta estructura.

Esta noción es la que describe exactamente el número de nivel que se asocia a cada unidad. Para aclarar volvamos a describir el archivo y su registro dando a cada unidad un nombre y un número de nivel.

```
DATA  DIVISION.
FILE  SECTION.
FD    ARCHIVO-PIEZAS ----.
01    PIEZA ----.
 02   IDENTIFICACION ----.
  03  CLAVE ----.
  03  FABRICANTE ----.
  03  FECHA-FABRICACION ----.
 02   CARACTERISTICAS-TECNICAS ----.
  03  DIMENSIONES ----.
      07 ALTO ----.
      07 ANCHO ----.
      07 LARGO ----.
  03  DUREZA ----.
 02   CONTROL-ALMACEN ----.
  03  FECHA-RECEPCION ----.
  03  FECHA-VENTA ----.
  03  EXISTENCIAS ----.
  03  PUNTO-PEDIDO ----.
```

En esta descripción '-----' indica elementos de la descripción todavía sin definir.

Los números de nivel son 01 a 49. FD se utiliza sola para las entradas de descripción de archivos. Números de nivel más alto indican subdivisiones de los más bajos. Elementos del mismo orden deben de tener el mismo número de nivel. Como se ve en el grupo DIMENSION y sus elementos ALTO, ANCHO, y LARGO, los números de nivel no tienen que ser correlativos, basta con que las divisiones tengan el número mayor. Los nueve primeros números 01, 02, .., 09 pueden escribirse también 1, 2, ..., 9.

FD y 01 tienen que escribirse a partir del margen A. Los demás pueden sangrarse a la derecha para resaltar la estructura. Todos los nombres en cambio pueden iniciarse en el margen B o sangrarse a la derecha de este margen. Entre el número de nivel y el nombre debe de haber por lo menos un blanco.

Además de estos números de nivel existen otros dos números con significado especial, estos son el 77 y 88.

El 77 se utiliza para nombres de dato independientes, datos que no pertenecen a ningún registro. Por esta causa este nivel sólo se usa en la WORKING-STORAGE y CONSTANT SECTION. El número 77 precede a todos los demás niveles y se escribe siempre a partir del margen A. El número 88 se utiliza para los nombres de condición, se utiliza en la FILE SECTION y en la WORKING-STORAGE, se escribe a partir del margen A donde se desee.

La WORKING-STORAGE y la CONSTANT SECTION se estructuran como la FILE SECTION pero no existen descripciones de archivo (FD). En cambio en estas secciones hay entradas de nivel 77 que no existen en la FILE SECTION.

5. - Calificación

Es frecuente en un programa que haya más de un registro con elementos o grupos con nombre repetidos. Para poder identificar cada uno de los elementos o grupos se utiliza la calificación que consiste en seguir al nombre con las partículas OF o IN y el nombre del grupo inmediatamente superior y así hasta que quede completo.

Ejemplos:

```
PRECIO OF ARTICULO IN VENTAS
PRECIO OF ARTICULO IN COMPRAS
SUELDO OF COLABORADOR IN CONTRATO-1
SUELDO IN COLABORADOR OF CONTRATO-1
```

La calificación también se utiliza en la PROCEDURE para identificar párrafos con el mismo nombre en secciones distintas.

Ejemplos:

```
GO TO 2 IN SEGUNDO
```

Como resumen final vamos a describir la DATA DIVISION de la aplicación del Tema VI.

```
DATA  DIVISION.
FILE  SECTION.
FD    ARCHIVO-GENERAL-V LABEL RECORDS ARE STANDARD
      DATA RECORD IS LIBRO.
01    LIBRO.
 02     NUMERO-CLAVE PICTURE X (6).
 02     TITULO PICTURE X (15).
 02     AUTOR PICTURE A (12).
 02     MATERIA PICTURE A (6).
 02     EDITORIAL PIC A (15).
 02     PRECIO-COMPRA PIC 9 (5) COMP.
 02     PRECIO-FUERTE PIC 9 (5) COMP.
 02     NUMERO-EJEMPLARES PIC 99 COMP.
FD.   ARCHIVO-GENERAL-N LABEL RECORDS ARE STANDARD
      DATA RECORD IS LIBRO-N.
01    LIBRO-N.
 02     NUMERO-CLAVE PIC X (6).
 02     AUTOR PIC A (12).
 02     MATERIA PIC A (6).
 02     EDITORIAL PIC A (15).
 02     PRECIO-COMPRA PIC 9 (5), COMP.
 02     PRECIO-FUERTE PIC 9 (5), COMP.
 02     NUMERO-EJEMPLARES PIC 99, COMP.
FD    MOVIMIENTO LABEL RECORDS ARE OMITTED
      DATA RECORD IS LIBRO-ACTUALIZADO.
01    LIBRO-ACTUALIZADO.
```

```
 02    NUMERO-CLAVE PICTURE X (6).
 02    FILLER PIC X (68).
 02    FECHA PIC X (6).
FD     VENTAS LABEL RECORDS ARE OMITTED DATA RECORD
       IS LINEA.
01     LINEA PIC X (128).
WORKING-STORAGE SECTION.
77     TOTAL PIC 9 (6) COMP.
01     LIBRO-VENDIDO.
 02    NUMERO-CLAVE PIC X (6).
 02    TITULO PIC X (15).
 02    FILLER PIC XX CALUE SPACES.
 02    AUTOR PIC A (12).
 02    FILLER PIC XX VALUE SPACES.
 02    MATERIA PIC A (6).
 02    FILLER PIC XX VALUE SPACES.
 02    EDITORIAL PIC A (15).
 02    FILLER PIC XX VALUE SPACES.
 02    PRECIO-FUERTE PIC ZZZ,ZZ9.
01     L-TOTAL.
 02    FILLER PIC X (55) VALUE SPACES.
 02    FILLER PIC X (9) VALUE 'TOTAL... '.
 02    TOTAL-EDIT PIC ZZZ,ZZ9.
PROCEDURE DIVISION.
```

El alumno tiene que contentarse por el momento con examinar la estructura de las entradas. Los dos temas próximos darán información suficiente para una comprensión total.

TEMA VIII

ENTRADAS DE DESCRIPCION DE REGISTROS

1.- Elementos de una entrada

Los registros se describen por medio de entradas, que definen tanto a ellos mismos como a sus componentes grupos y elementos. Una entrada consiste en un número de nivel, un nombre una serie de cláusulas (la serie puede ser nula) terminadas con un punto.

Damos a continuación el formato de entrada que vamos a estudiar. Es un formato reducido, COBOL tiene muchas más cláusulas, pero las dadas permiten describir de una forma completa cualquier registro.

número-nivel { FILLER / nombre-dato-1 } [; { PIC / PICTURE } IS caracteres]

[; USAGE IS { COMP / COMP-1 / DISPLAY / INDEX }]

[; OCCURS entero TIMES]

[; VALUE IS literal]

2.- PIC, USAGE, OCCURS y VALUE son cláusulas. Por medio de cláusulas se dan las características que posee un nombre de dato; longitud, tipo, uso e incluso valor.

Obsérvese que detallando plenamente cada uno de los elementos que componen un registro, éste queda definido totalmente. Por este motivo a nivel de registro e incluso de grupo no es necesario incluir cláusula ninguna. Pero si se incluye alguna esto no puede contradecir a las que describen a cada uno de sus elementos.

3.- Cláusula USAGE

Cada nombre de dato puede usarse con dos fines:para cálculo o para representación. En el archivo de la librería (enla aplicación ejemplo de los temas anteriores), el nombre del autor, el título de la obra, etcétera, son utilizados sólo con fines de representación (DISPLAY) que en algún momento has de escribir en un informe. El importe del precio fuerte sería en cambio para irlo sumando al nombre TOTAL para obtener el importe de las ventas: estos usos son de cálculo (COMP).

Hay otra cuestión a analizar de gran importancia, la representación interna de estas unidades de información. Sin embargo siendo este particular totalmente dependiente del ordenador, sólo podemos tratarles de una forma bastante superficial.

En la mayor parte de los ordenadores, la memoria se compone de palabras que a su vez contienen un número determinado de bits. Esta noción es cercana a la de bute que es la menor unidad referenciable por programa. Si una palabra cumple esta condición, la palabra es un byte en ese ordenador, pero otras veces la palabra contiene cuatro o seis bytes (u otras cantidades). Un byte suele tener seis u ocho bits.

Ocho bits admiten 256 códigos, seis bits admite 64 códigos. Por lo tanto cualquiera de estos bytes es capaz de admitir la codificación de todos caracteres COBOL. Pero para representar los números del 0 al 9 sobran cuatro bits (16 códigos). Esto hace que si definimos un campo numérico éste se puede empaquetar ocupando menos memoria. Para fijar ideas a un nombre-dato usado para cálculo pero sin empaquetar le podríamos definir COMP y a un nombre-dato empaquetado le podríamos llamar COMP-1 (COMPUTATIONAL y COMPUTATIONAL-1 son nombres equivalentes a ellos).

Por último debemos recordar que los ordenadores realizan sus operaciones aritméticas empleando palabras enteras cuyo valor es el equivalente decimal de su representación en bits (equivalente-entero). Las operaciones con nombres-dato de las características definidas en la DATA DIVISION se realizan por medio de subrutinas que genera el com-

pilador o que éste incluye en el programa traducido. La operación de sumar ordenada por ADD A TO B, es por lo tanto mucho más compleja que su equivalente en código máquina, que podrían describir así:

```
LD      A     carga A es el acumulador
ADD     B     sumar B al contenido del acumulador
STO ,   B     almacenar en B.
```

El motivo por el que no se traducen a esta simple secuencia de operaciones es que los números que se manejan en COBOL pueden ser bastante más grandes que los que puede contener una palabra y además la mayor parte de las veces no son enteros. Pero si se dan estas condiciones sería lamentable el utilizar subrutinas cuando una secuencia tan breve y efectiva como la anterior sería suficiente. Pues bien esto lo puede realizar el compilador siempre que tenga la información adecuada: ésta puede ser la palabra INDEX. Esta resulta una palabra adecuadapor que los índices cumplen las cualidades expuestas y por lo repetido de su proceso es muy conveniente que sea lo más rápido posible.

Cuando no se indica nada, no existe la cláusula USAGE, el ordenador supone que su uso es DISPLAY. Nombres-dato definidos COMP pueden usarse en operaciones DISPLAY y viceversa, la conversión de un formato a otro la efectúa el compilador, sólo la eficiencia del programa puede resentirse de estas faltas de definición.

A un nombre-dato que ha de contener caracteresque no son números no se le puede definir COMP. A un grupo se le puede definir COMP siempre que sus elementos lo sean, pero no se pueden realizar operaciones aritméticas a nivel de grupo.

4.- PICTURE o PIC

Esto es la cláusula más importante, su potencia hace que otras muchas cláusulas COBOL sean innecesarias o se usen poco.

Para que la presentación sea más sistemática y clara, vamos a clasificar los datos así:

```
AB  = alfabéticos
ABE = alfabéticos editados
AN  = alfanuméricos
ANE = alfanuméricos editados
N   = numéricos
NE  = numéricos editados
```

4.1. Datos alfabéticos (AB)

Son alfabéticos los que contienen las letras A a la Z y el blanco. Se representan con letras A así:

02 TITULO PIC AAAAAAA

El nombre de dato TITULO contiene siete letras o espacios

LETRAS, AGUA, PEPE

serían contenidos válidos de este campo.

AGUAVIVA en cambio no cabe.

Para evitar la repetición de la letra A (y lo mismo ocurre con otros signos que se introducirán después) se puede colocar detrás de ella y entre paréntesis el número de veces que se repite.

02 TITULO PIC A (7)

4.2. Datos alfabéticos editados (AE)

El concepto de edición es tan fundamental en COBOL que se dice que el COBOL no sería el COBOL sin la edición.

La edición es una transformación efectuada en una secuencia de caracteres contenidos por un campo emisor, definida por unos caracteres clave contenidos por el campo receptor.

Los campos alfabéticos sólo tienen un carácter de edición: la letra B. Esta letra indica que en su lugar ha de aparecer un blanco.

Sean:

05 NOMBRE PIC A (8) NOMBRE contiene AGUAVIVA
05 NOM -E PIC A (4) B A (4)

después de un

MOVE NOMBRE TO NOM-E

el nombre-dato NOM-E contendrá AGUA VIVA

4.3. Datos numéricos (N)

Para representar los datos numéricos se utilizan las letras: S, V, P y 9.

9 indica un número del 0 al 9 y puede repetirse cuanto sea necesario. NOTA.- En todas las descripciones de PIC que hacemos el número de caracteres que siguen a PIC no puede ser más de 30. En PIC A (6) esta longitud es 4.

Ejemplo:

04 TELEFONO PIC 9 (7).

admite números de teléfonos con siete cifras.

Es evidente que el uso que damos a este dato es DISPLAY, en cambio en

07 PRECIO PIC 999 V99 COMP.

usar el precio con fines de cálculo es natural.

Estos ejemplos deben dejar claro que el describir un dato con nueves, indica que el campo es numérico (contiene números), pero no que su uso sea COMPUTATIONAL.

La V indica la posición de la coma decimal, sin embargo esta coma no se almacena. Consideremos que el nombre-dato anterior (PRECIO) tiene el valor 86.32, en memoria estará almacenado así:

0	8	6	3	2

ocupa cinco lugares, que corresponden a otros tantos nueves de la PIC. Sin embargo la posición de la coma queda registrada y en cualquier operación que se haga con este dato, el valor será 86.32.

Ejemplos:

MOVE PRECIO TO TEMPORAL

Veamos los resultados según distintos PIC en TEMPORAL

TEMPORAL

PIC 9 (4) V9	0 0 8 6 3
PIC 99 V (9)	8 6 3
PIC 999	0 8 6
PIC 9V999	6 3 2 0

Estudiando estos ejemplos se verá claramente cómo la V de los campos fuente y receptor determinan la forma de la transmisión, de derecha a izquierda a la izquierda del punto decimal; de izquierda a derecha, a la derecha del punto decimal. Cuando sobra capacidad en el campo receptor se rellena de ceros las posiciones vacantes, cuando falta espacio, la transmisión se trunca (el compilador enviará un mensaje de advertencia). Esto ocurre a la izquierda y a la derecha del punto decimal simbólico.

Cuando no existe V (ejemplo: PIC 99), la coma se pone en el extremo derecho. Pero si la coma no está aquí sino más posiciones a la derecha, entonces empleamos la letra P que indica esta posición.

Ejemplo: 22 000 000 quedarían en memoria con un PIC 99P(6) así: | 2 | 2 | con un PIC 99 solo entraría | 0 | 0 |

También las P se pueden colocar a la izquierda, indicando también la posición del punto decimal.

Ejemplo: PIC PP 99 permitiría representar el número 0.0064 como | 6 | 4 |

Por último el signo se indica con la letra S. Cuando no se pone la letra S todos los números positivos o negativos se consideran positivos:

Ejemplo:

```
MOVE SALDO TO IMPORTE
SALDO  PIC S999V99    9999V9
Si SALDO es -86.35  después de la operación
   IMPORTE contiene | 0 | 0 | 8 | 6 | 3 |
```

El signo se presenta de forma distinta en cada ordenador y ocupa un lugar (de formas distintas) en la memoria. Sin embargo en COBOL no se le cuenta para determinar la longitud de un nombre-dato. Se le suele representar cuando es menos con un guión encima de la última posición a la derecha. El contenido de SALDO según este criterio sería:

| 0 | 8 | 6 | 3 | 5 |

4.4. Datos numéricos editados (NE). Primero opción.

La edición en este caso es la intercalación de ceros en el campo receptor y es del todo equivalente a la edición de datos alfabéticos. El símbolo clave es el número 0.

Ejemplo:

MOVE TELEFONO TO TELEFONO-E'
Si TELEFONO-E es PIC 9099099099
y TELEFONO contenía [2|7|6|5|8|0|0]

TELEFONO-E contendría 2076058000

4.5. Datos alfanuméricos (AN)

Estos campos pueden contener cualquier carácter válido COBOL. Se representan con los caracteres A, X y 9. La A y el 9 representan letras o números respectivamente. La X representa cualquier carácter.

Ejemplos: 06 CLAVE PIC AX999.

describiría una clave que contuviese identificaciones así: E-064, H-127, etcétera.

También podría haberse descrito así: PIC X(5).

4.6. Datos alfanuméricos editados (ANE)

Los caracteres de edición son el 0 y B con el mismo significado que en los numéricos y alfabéticos respectivamente.

Ejemplo:

MOVE TELEFONO TO TELEFONO-E
TELEFONO-E PIC XBXXBXXBXX

después de la operación TELEFONO-E contendría [2 76 58 00]

4.7. Antes de introducir la segunda opción de los números editados(NE), vamos a examinar cómo se realizan las comparaciones.

Recordemos que una expresión condicional tenía el formato:

{ nombre-datos-1 / expresión-aritmética-1 / constante-figurativa-1 / literal-1 } IS NOT relación { nombre-datos-2 / expresión-aritmética-2 / constante-figurativa / literal-2 }

Hasta ahora nos hemos contentado con saber que una expresión condicional como ésta sólo puede tener uno de los dos valores posibles; verdadero o falso. Veamos como lo determina el programa compilado.

En los ejemplos supondremos la relación < , pero sería semejante con cualquiera otra de las tres relaciones.

(1) nombre-dato-1 < nombre-dato-2

Estudiándolo según la clase de información que contengan.

n-d.-1	n.d.-2	
AB	AB	Se comparan los caracteres de izquierda a derecha, la primera pareja desigual nos indicará el sentido de la desigualdad (los valores son los de la secuencia de intercalación). En caso de que los caracteres comparados sean iguales y uno de los datos termine el otro sería mayor a no ser que el resto de sus caracteres sean blancos. De esta forma determinamos si la expresión condicional es verdadera o falsa.
AB	ABE	igual
AB	AN	" (AN solo letras y blancos)
AB	ANE	" "
AB	N	no se comparan
AB	NE	" "
AN	ANE	como AB : AB
AN	N	" "
AN	NE	" "
ANE	ANE	" "
ANE	N	" "
ANE	NE	" "
N	N	como AB : AB si son DISPLAY comparando sus valores numéricos si son COMP.
N	NE	como AB : AB

(2) nombre-dato-1 < expresión-aritmética-2

Como expresión-aritmética-2 es N y COMPUTATIONAL, esta comparación será como una de las analizadas según sea nombre-dato-1.

(3) nombre-dato-1 < constante-figurativa

Las constantes figurativas son ZERO y SPACE, como ZERO es N y SPACE X o A también este caso se ha estudiado.

(4) nombre-dato-1 < literal

Literal puede ser numérico (N) o no numérico (A ó X) en cualquiera de estos casos lo hemos estudiado.

(5) expresión-aritmética < expresión-aritmética

Las dos son N y COMPUTATIONAL, por lo tanto se comparan sus valores numéricos, caso ya estudiado.

El alumno puede proseguir el análisis y convencerse de que todos los casos posibles se reducen a los analizados.

4.8. Numéricos editados (NE). Segunda opción.

En la definición de numéricos editados se pueden utilizar los caracteres de edición . , + , −, Z, *, $, CR y DB.

(1) Uso de .

MOVE IMPORTE TO IMPORTE-EDITADO
IMPORTE PIC 999V99 valía 16.64
IMPORTE-EDITADO PIC 999.99
contendría | 0 | 1 | 6 | . | 6 | 4 | que es un campo AN

El . aparece en el lugar que le corresponde (como indica la V) y ocupa lugar en memoria.

(2) Uso de la ,

En esta y en las siguientes explicaciones usaremos el MOVE de (1).

IMPORTE PIC 9 (4)V99 valía 1264.65
IMPORTE-EDITADO 9,999.99 contendría [1, 264.65]

La , como el . ocupan la posición que les corresponde.

(3) Uso de +

IMPORTE PIC S9(4)V99 valía 82.64
IMPORTE-EDITADO PIC + 9,999.99 contendría [+ 0 , 0 8 2 . 6 4]

IMPORTE-EDITADO PIC + + , + + 9.99 contendría [+ 82.64]

Obsérvese la diferencia entre el uso de un + y una cadena de signos +

IMPORTE PIC S9(4)V99 valía -82.64
IMPORTE-EDITADO PIC + + , + + 9.99 contendrá [- 8 2 . 6 4]

IMPORTE-EDITADO PIC 999.999 + [0 8 2 . 6 4 -]

(4) Uso de -

IMPORTE PIC S9(4)V99 valía 82.64
IMPORTE-EDITADO PIC -9,999.99 contendrá [0 , 0 8 2 . 6 4]

IMPORTE-EDITADO PIC - - , - - 9.99 " [- - - 8 2 . 6 4]

IMPORTE-EDITADO PIC 999.99- [0 8 2 . 6 4]

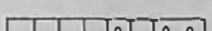
IMPORTE PIC S9(4)V99 valía -82.64
IMPORTE-EDITADO PIC 999.99- contendría [0 8 2 . 6 4 -]

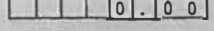
(5) Uso de Z

IMPORTE PIC S9(4) V99 valía 82.64
IMPORTE-EDITADO PIC - Z, ZZ9.99 contendría [8 2 . 6 4]

IMPORTE-EDITADO PIC + Z, ZZ9.99 " [+ 8 2 . 6 4]

IMPORTE PIC S9(4)V99 valía -82.64
IMPORTE-EDITADO PIC Z,ZZ9.99- contendría 

IMPORTE PIC S9(4)V99 valía 0.0
IMPORTE-EDITADO PIC Z,ZZ9.99 contendría 

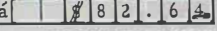
IMPORTE-EDITADO PIC Z,ZZZ,ZZ " 

(6) Uso de $

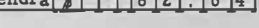
IMPORTE PIC S9(4)V99 valía -82.64
IMPORTE-EDITADO PIC $, $$$. 99 contendrá 

IMPORTE-EDITADO PIC $Z,ZZ9.99- contendrá 

(7) Uso de CR y DB

IMPORTE PIC S9(4)V99 valía -82.64
IMPORTE-EDITADO PIC *, ***.** - 

IMPORTE-EDITADO PIC *, **9.99 CR 

IMPORTE-EDITADO PIC *, **9.99DB 

Normas

a.- El +, -, y $ pueden utilizarse una vez o formando cadenas flotantes.

Se llama cadena flotante a series de signos iguales tal como - - , + + + . + + entre los que se pueden incluir otros como la , y el .

Cuando se utilizan una vez, se colocan delante de la serie de caracteres.

b.- Cuando se pone el +, si el número es positivo aparece el signo + y el + cuando es negativo.

Cuando se pone el -, si el número es positivo aparece un blanco y el signo - cuando es negativo.

c.- El signo * no puede aparecer una sola vez (siempre forma cadena).

d.- La Z indica impresión supresión simple de ceros a la izquierda.

e.- C R y D B se usan igual que el signo - colocado al final de PIC - TURE.

f.- Los NE no pueden utilizarse en operaciones aritméticas. En reali - dad son AN y se comportan como tales en comparaciones y con el verbo MOVE. Tampoco son COMPUTATIONAL pues su uso es DIS- PLAY. En efecto el fin de la edición es que cuando estos datos se impriman aparezcan en forma legible.

Para terminar con la cláusula PICTURE señalaremos que ésta des cribe plenamente el formato y la clave de contenido de los nombres-dato. Esta cláusula sólo puede incluirse en la descripción de elementos, nunca a nivel de grupo.

5.- La cláusula VALUE

Opción 1.

[; VALUE IS literal]

Ejemplos:

02 P VALUE IS 126.84 PIC 999V99.
06 TITULO VALUE IS 'RESUMEN ZONAS' PIC A (13)

Se usa para definir nombres-dato con un valor constante (CONS - TANT SECTION) o literales para imprimir. También se utiliza para ini ciar los datos con un valor determinado cuando se inicia el programa. Muy corriente es la siguiente entrada en la WORKING STORAGE.

03 SUMA PIC 9 (5)V99 VALUE ZEROS.

con lo que aseguramos que el campo SUMA contiene cero cuando se ini- cia el cálculo.

Opción 2.

[; {VALUE IS / VALUES ARE} literal-1 [THRU literal-2] [literal-3 [THRU literal-4] ...]]

Esta opción se usa sólo con nombres de condición. Veamos unos ejemplos:

```
05   NACIDO PIC  AA.
     88 MADRID VALUE 'MD'
     88 BARCELONA VALUE 'BR'
     88 NO-OTROS VALUE 'MD', 'BR'
```

Y podríamos escribir declaraciones así:

```
IF MADRID PERFORM CALCULO-1.
IF NOT NO-OTROS GO TO CALCULO-SUCURSALES.
```

Otro ejemplo:

```
03   RESULTADO-EXAMEN PIC 99.
  88   SUEPENDIDO VALUE O THRU 4.
  88   APROBADO VALUE 5 THRU 6.
  88   NOTABLE VALUE 7 THRU 8.
  88   SOBRESALIENTE VALUE 9 THRU 10.
```

Y en la PROCEDURE podríamos hacer declaraciones como ésta:

```
IF NOTABLE PERFORM HONORES
```

6. - La cláusula OCCURS

Su formato simple es:

[; OCCURS entero TIMES]

Ejemplo:

```
03 PARTIDA PIC 9(5) OCCURS 14 TIMES.
```

que indica que el nombre PARTIDA aparece 14 veces ocupando un total de 80 posiciones numéricas.

La cláusula OCCURS puede aparecer a cualquiernivel menos el 01, 77 y 78 lo que recordando el significado especial de estos niveles resulta evidente.

Para referirnos en la PROCEDURE a uno determinado de estos elementos se indica por medio de un índice que se pone entre paréntesis así:

PARTIDA (5) , PARTIDA (2), etc.

Si el nombre está calificado se pone así:

PARTIDA OF GRUPO-2 IN MATRIZ (5)

Una cláusula OCCURS puede aparecer en un grupo que su vez contenga otro con otra cláusula OCCURS y aún éste con otra cláusula OCCURS como máximo.

Ejemplos:

```
01    TABLA.
  02  ESTADO OCCURS 48.
      03 PROVINCIA OCCURS 26.
         07 MUNICIPIO OCCURS 18, PIC X (72).
```

Para referirnos al estado número 7 diremos:

ESTADO (7)

Para referirnos a la provincia número 21 diremos:

PROVINCIA (I, 21)

donde I indica el estado. I es un nombre dato índice.

Para referirnos a la provincia 21 del estado 42 diremos

PROVINCIA (42, 21)

Para referirnos al municipio 14 de la provincia 5 del estado 14 diversos

MUNICIPIO (14, 5, 14)

Con un MOVE TABLA TO CAMPO

trasladaríamos toda la tabla al nombre-dato llamado CAMPO que para contenerla entera tiene que tener una capacidad de 1517408 caracteres, dimensión bastante mayor que las memorias usuales.

Con un MOVE ESTADO (5) TO CAMPO trasladaríamos todo el estado 5 a CAMPO que debería tener una capacidad de 33 696 caracteres.

Con un MOVE PROVINCIA (5, 17) TO CAMPO

trasladaríamos toda la provincia 17 del estado 5 a CAMPO que debería tener una capacidad de 1296 caracteres.

Por último con un

MOVE MUNICIPIO (5, 12, 17) TO CAMPO

trasladaremos el municipio 17 de la provincia 12 del estado 5 a CAMPO que deberá tener una capacidad de 72 caracteres.

Operaciones similares podríamos efectuar con los verbos READ y WRITE.

Ejemplo:

```
READ FICHA INTO MUNICIPIO (5, 12, 17)
WRITE REGISTRO FROM PROVINCIA (5, 12)
```

En cambio no podríamos trasladar directamente todos los municipios 14 a CAMPO.

Esto lo podríamos hacer así:

```
DATA DIVISION
  .
  .
  .
WORKING- STORAGE SECTION.
77   I PIC 99, INDEX.
77   J PIC 99, INDEX.
77   K PIC 9 (4), INDEX.
  .
  .
  .
01   CAMPO.
  02   ITEM OCCURS (1248) PIC X (72).
  .
  .
```

```
PROCEDURE DIVISION.
    .
    .
    .
        MOVE ZERO TO K
        PERFORM TRASLADO VARYING I FROM 1 BY 1
        UNTIL I > 48  AFTER J FROM 1 BY 1 UNTIL J > 26
    .
    .
    .
TRASLADO.
        K = K + 1
        MOVE MUNICIPIO (I, J, 14) TO ITEM (K).
```

Como resumen final y aplicación de la cláusula OCCURS y el verbo PERFORM con la opción VARYING vamos a realizar una serie de operaciones elementales con matrices.

Sea la matriz:

```
01   MATRIZ.
 02    FILAS OCCURS 6 TIMES.
        03 COLUM OCCURS 7 PIC 999.
```

La matriz nula la obtendremos con

```
        MOVE ZEROS TO MATRIZ
```

Sea ahora la matriz cuadrada:

```
01   MATRIZ.
 02    FILAS OCCURS 6 TIMES.
        03 COLUM OCCURS 6 TIMES PIC 999.
```

y las operaciones

```
        MOVE ZEROS TO MATRIZ
        PERFORM UNIDAD VARYING I FROM 1 BY 1 UNTIL I > 6
            .
            .
            .
UNIDAD.
        MOVE 1 TO COLUM (I, I).
```

que la convierte en matriz unidad.

Sean las matrices:

```
01  MATRIZ-1.
    02  FILAS-1 OCCURS 6 TIMES.
        03 COLUM-1 OCCURS 7 TIMES PIC 999.
01  MATRIZ-2.
    02  FILAS-2 OCCURS 6 TIMES.
        03 COLUM-2 OCCURS 7 TIMES PIC 999.

Su  suma se obtendrá así
        PERFORM SUMA VARYING I FROM 1 BY 1 UNTIL I > 6
                     AFTER J  FROM 1 BY 1 UNTIL J > 7.
.
.
.
SUMA.  ADD COLUM-1 (I, J) TO COLUM-2 (I, J).
```

Para multiplicar dos matrices es necesario que el número de columnas de la primera sea igual al de filas de la segunda. El resultado aparecerá en una tercera matriz que tendrá el número de filas de la primera matriz y el de columnas de la segunda.

Sean las matrices:

```
01  M-1.
    02  F-1 OCCURS 16 TIMES.
        03 C-1 OCCURS 12 TIMES PIC 999V99.
01  M-2.
    02  F-2 OCCURS 12 TIMES.
        03 C-2 OCCURS 18 TIMES PIC 999V99.
01  MATRIZ-PRODUCTO.
    02  F OCCURS 16 TIMES.
        03 C OCCURS 18 TIMES PIC 9(6)V99.

El  producto de M-1 por M-2 se obtendrá así:
        MOVE ZEROS TO MATRIZ-PRODUCTO
        PERFORM PRODUCTO VARYING I FROM 1 BY 1 UNTIL I > 16
                              AFTER  J FROM 1 BY 1 UNTIL J > 18
                              AFTER  K FROM 1 BY 1 UNTIL K > 12
.
.
.
PRODUCTO.
        COMPUTE C (I, J) = C (I, J) + C-1 (I, K) * C-2 (K, J).
```

TEMA IX

CONSTRUCCION DE LA DIVISION DE DATOS

1.- Sección de archivos

Esta sección recibe el nombre de FILE SECTION y es la primera de las tres secciones de la DATA DIVISION que estudiamos en la primera parte del curso. Su fin es la descripción de los archivos que se utilizan en el proceso.

En COBOL "archivo" es una palabra técnica cuyo significado es en muchas aplicaciones muy distinto del uso normal de esta palabra.

Un archivo está formado por registros, que estos registros contengan información homogénea, aunque frecuente, es en realidad indiferente para que constituyan un archivo. Para que en COBOL unos registros constituyan un archivo, sólo es necesario que lo definamos así en el programa.

Los registros pueden ser diferentes en estructura tanto por la división en campos como por tener longitudes diferentes, y pueden ser diferentes en contenido. Un archivo puede contener registros de uno, dos o más tipos.

Vamos a poner una serie de ejemplos sencillos que aclararán estos conceptos.

1.- Un archivo de personal está perforado en fichas (en COBOL una ficha es un registro). Toda la información de un empleado se perfora en tres fichas (tres registros), por lo tanto si la empresa tiene mil empleados, el archivo se compone de tres mil fichas. (En estos casos al archivo se le suele llamar fichero.)

Las fichas vendrán perforadas así:

Ficha 1

(1, 6) Clave de empleado
(7, 7) Clave de alta o baja
(8, 22) Nombre
(23, 48) Apellidos
(48, 62) Domicilio
(63, 69) Teléfono
(70, 76) DNI
(77, 79) Clave de departamento
(80, 80) Número de ficha que en esta es 1.

Ficha 2

(1, 6) Clave de empleado
(7, 16) Número de seguros sociales
(17, 18) Clave de categoría laboral
(19, 42) Nombre de categoría laboral
(43, 48) Fecha de nacimiento
(49, 56) Fecha de ingreso en la empresa
(57, 62) Fecha del título profesional
(63, 68) Carnet de familia numerosa
(69, 69) Categoría del carnet
(70, 70) Estado civil
(71, 72) Número de hijos
(73, 74) Claves de ayuda familiar
(75, 79) Subsidio familiar
(80, 80) Número de ficha, que en esta es 2

Ficha 3

(1, 6) Clave de empleado
(7, 13) Sueldo anual
(14, 20) Remuneración complementaria
(21, 27) Primas anuales fijas
(28, 32) Importe de horas extras
(33, 34) Tarifa seguros sociales
(35, 36) Claves seguros sociales
(37, 38) Claves de utilidades
(39, 45) Anticipos
(46, 50) Otros descuentos
(51, 57) Sueldos percibidos en el año
(58, 64) Remuneraciones percibidas en el año

(65, 71) Primas percibidas en el año
(72, 78) Horas extras percibidas en el año
(79, 79) No utilizado
(80, 80) Número de la ficha, que en esta es 3.

Nota: en (17, 32), 17 indica la primera columna del campo y 32 la última. Se suponen fichas de 80 columnas.

En este ejemplo se puede ver un archivo compuesto por tres tipos de registros distintos por su estructura y contenido.

2.- El archivo que ahora presentamos es un informe impreso de las ventas de un almacén de materiales. En este informe las ventas vendrán clasificadas por zonas, y dentro de cada zona vendrán primero las ventas directas, después las ventas a través de representantes y por último una línea de total.

Un fragmento de este informe sería el siguiente:

VENTAS DIRECTAS

CLAVE	DESCRIPCION	UNIDAD	PRECIO	IMPORTE
X-22-3	AGUJAS ACERO	23	12,80	294,40
X-26-5	AGUJAS ACERO	12	6,80	81,60
A-63-0	PINZAS MED.	63	20,00	1260,00
B-65-4	PRENSOR	6	264,00	1584,00
			SUMA	3220,00

VENTAS POR REPRESENTANTES

CLAVE	DESCRIPCION	REP.	UNIDAD	PRECIO	IMPORTE
X-22-3	AGUJAS ACERO	2C	17	12,80	217,60
A-63-1	PINZAS NORM.	4C	15	200,00	3000,00
A-63-5	PINZAS LAB.	5C	8	150,00	1200,00
				SUMA	4417,60
			TOTAL ZONA CENTRAL		7637,60

En este archivo hay ocho tipos de registros (recuérdese que una línea de impresión es un registro), que corresponden cuatro a las cabeceras:
VENTAS DIRECTAS, CLAVE DESCRIPCION UNIDAD etc. y CLAVE DESCRIPCION REP UNIDAD etc.

Dos de resúmenes SUMA importe y TOTAL ZONA zona importe y dos de detalle que podrían describirse así:

Línea de detalle 1

(1, 6) Clave
(8, 20) Descripción
(25, 27) Unidad
(29, 38) Precio
(40, 49) Importe

Línea de detalle 2

(1, 6) Clave
(8, 20) Descripción
(22, 20) Representante
(25, 27) Unidad
(29, 38) Precio
(40, 49) Importe

Estos registros son en realidad de longitud variable, además de ser distintos en contenido y estructura. Sin embargo por ser la línea de una impresora de longitud máxima fija (132 caracteres por ejemplo) se les suele describir como de longitud fija.

3.- En este ejemplo consideraremos un archivo en banda magnética conteniendo la información sobre los libros existentes a la venta en una librería. Sus registros tendrían la siguiente composición:

(1, 6) Clave
(7, 26) Título
(27, 56) Materia
(57, 70) Editorial
(71, 80) Nacionalidad
(81, 87) Precio
(88, 91) Número de ejemplares

En este caso todos los registros son idénticos por su contenido, estructura y longitud.

Con los tres ejemplos presentados quedan señaladas aquellas peculiaridades que deben ser descritas en la FILE SECTION. Sólo debemos advertir que las diferencias resultantes no se deben sólo a los me-

dios utilizados: fichas, líneas de impresión, etc.; sino que por ejemplo el registro en banda magnética puede tener todas estas características.

También señalaremos que estas descripciones no sólo pueden hacerse en la FILE SECTION, sino que pueden hacerse también en la WORKING-STORAGE y usarlos combinadamente como veremos al tratar esta última sección.

En un programa pueden utilizarse uno o más archivos, todos ellos han de describirse por separado en la FILE SECTION.

La descripción de un archivo se hace empezando por hacer una descripción general del mismo y de los registros que le componen, seguido del detalle de estos registros.

Un formato simplificado de la entrada de descripción de un archivo es la siguiente:

```
A   | B
FD  | nombre de archivo [BLOCK CONTAINS entero RECORDS]
    |   ; LABEL RECORD IS {STANDARD}
    |                     {OMITTED}
    |   ;[DATA {RECORD  IS}      nombre-dato-1
    |          {RECORDS ARE}
    |    [nombre-dato-2 ...]]
```

Pasemos a analizar las cláusulas que componen el formato:

```
[BLOCK CONTAINS entero RECORDS]
```

Ejemplos: BLOCK CONTAINS 3 RECORDS
BLOCK 3 RECORDS

Se usa con las bandas magnéticas. Estas se leen y graban por bloques y un bloque puede contener uno o más registros.

Entre bloque y bloque tiene que haber espacios fijos sin utilizar que permiten el acelerado y frenado de las guías que hacen girar los carretes que contienen las bandas. Por lo tanto cuantos más bloques existen (tienen menor tamaño) más longitud de banda se desperdicia. Teniendo esto en cuenta la longitud del bloque debería ser la mayor posible. Pero por otra parte, los bloques tienen que estar contenidos en memoria en el momento de grabar, y se reciben enteros cuando se leen. Por lo tanto si los bloques son muy grandes, o no caben en memoria o se desperdicia una parte demasiado grande. Esto hace necesario que los bloques sean estudiados cuidadosamente teniendo en cuenta las características especiales del ordenador. Incluso pueden existir otras características especiales (condiciones físicas) que determinen esta decisión.

```
[ ; LABEL RECORD IS { STANDARD }
                    { OMITTED  } ]
```

Sabemos que una banda magnética la monta el operador en uno de los armarios disponibles. Por medio de las fichas de control y las declaraciones de la ENVIORENMENT DIVISION se hace saber al programa cuál es este armario. ¿ Pero es la cinta montada la correcta? De no ser lo la lectura de una banda equivocada nos podría llevar a resultados incorrectos, por otra parte si grabamos en una banda equivocada podremos borrar información fundamental. El remedio es colocar en los archivos, delante del primer registro propio del archivo, un registro que contenga la identificación que deseamos del archivo. Entonces la primera operación a realizar será la comprobación de que el contenido de este registro especial (etiqueta) es el esperado. Como esta operación es normal efectuarla con archivos en banda magnética y en disco, los sistemas operativos tienen procedimientos típicos (STANDARD) que hace mención esta cláusula. Además hay otros métodos que todavía no estudiamos. Cuando no existen etiquetas, o el medio físico no las tenga: impresora, lectora de fichas, perforadora, etc., se pone la opción OMITTED.

```
[ ; DATA { REDORDS ARE } nombre-datos-1 nombre-datos-2... ]
         { RECORD IS   }
```

Esta cláusula permite relacionar la entrada de definición de archivo con los registros que le pertenecen. En el ejemplo del archivo formado por tres fichas del ejemplo anterior, podemos escribir:

```
DATA RECORDS ARE FICHA-1, FICHA-2, FICHA-3
```

A continuación de esta entrada de descripción de archivo se escriben las que describen sus registros. Como ejemplo vamos a describir completamente el archivo de fichas anterior, suponiendo ahora que está en banda

```
FD    ARCHIVO-EMPLEADOS BLOCK CONTAINS 3 RECORDS
      LABEL RECORD IS OMITTE DATA RECORDS ARE
      FICHA-1, FICHA-2,  FICHA-3

01    FICHA-1.
 02   CLAVE       PIC  X (6)
 02   SITUACION PICTURE A.
      88 ALTA VALUE 'A'.
      88 BAJA VALUE 'B'.
 02    IDENTIFICACION.
      03 NOMBRE PIC A  (15).
      03 APELLIDOS PIC A (25).
      03 DNI PIC  9(7).
       SEÑAS.
      03 DOMICILIO PIC  X (15).
      03 TELEFONO  PIC  9 (7).
 02    DEPARTAMENTO PIC  X(3).
 02    NUMERO PIC 9.

01    FICHA-2.
 02    CLAVE PIC  X(6).
 02    SITUACION-LABORAL.
      03 NUMERO-SS PIC 9(10).
      03 CAT-LABORAL PIC 99.
      03 NOM-LABORAL PIC X (24).
 02    FECHAS.
      03 NACIMIENTO PIC 9 (6).
      03 ING-EMPRESA PIC X (8).
      03 PROFESIONAL PIC  9 (6).
 02    SITUACION-FAMILIAR.
      03 CARNET-FN PIC 9 (6).
      03 CAT-FN PIC X.
      03 ESTADO CIVIL PIC A.
      03 HIJOS PIC 99.
      03 SUBSIDIO PIC 9 (5) COMP.
 02    NUMERO PIC 9.

01    FICHA-3.
 02    CLAVE PIC  X(6).
 02    INGRESOS.
```

```
        03 SUELDO PIC 9 (7) COMP.
        03 REMUNERACION PIC 9 (7) COMP.
        03 PRIMAS PIC 9 (5) COMP.
  02     SEGUROS.
        03 TARIFA PIC 99.
        03 CLAVES PIC 99.
  02     UTILIDADES PIC 99.
  02     DESCUENTOS.
        03 ANTICIPOS PIC 9 (7) COMP.
        03 OTROS PIC 9 (5) COMP.
  02     RESUMENES.
        03 SUELDOS PIC 9 (7) COMP.
        03 REMUNERACIONES PIC 9 (7) COMP.
        03 PRIMAS PIC 9 (7) COMP.
        03 HORAS PIC 9 (7) COMP.
  02     FILLER PIC X.
  02     NUMERO PIC 9.
```

En conclusión la FILE SECTION se compone de un conjunto de entradas de descripción de archivo seguidas de sus correspondientes descripciones de registro ordenadas así:

```
A       B
 DATA    DIVISION.
 FILE    SECTION.
 FD      (resto de la entrada).
 01      (entradas de registro).
   .
   .
   .
 FD      (resto de la entrada).
 01      (entradas de registro).
   .
   .
   .
```

Obsérvese que los nombres de división y sección terminan en punto, que FD y 01 empiezan en el margen A, los nombres ya sean de archivo, de registro, de grupo elemental, en B o a la derecha de B. Los números de nivel 02 en adelante pueden empezar en A o a la derecha de A.

2. - Sección de trabajo

En el programa hay que realizar una serie de operaciones con los datos leídos, con los literales y con las constantes definidas. Estas operaciones dan lugar a resultados parciales que sólo interesa conservar temporalmente y no se van a usar más una vez terminada la ejecución del programa. Otras veces ciertos resultados van a transmitirse a campos de registros de diversos archivos. En todas estas ocasiones se utilizan datos o registros definidos en la WORKING-STORAGESECTION.

En esta sección las entradas se hacen siguiendolas mismas reglas dadas para la descripción de registros, pero se diferencia de la FILE SECTION en que aquí no se definen archivos (FD-entradas) y en cambio existe un número de nivel (77) que no se utiliza en aquella.

El nivel 77 sirve para describir datos elementales independientes: datos que no son elementos de un grupo, y son elementales. Es muy usado este nivel para definir índices.

Ejemplo:

```
A     B
77    I  PIC 9 USAGE INDEX.
77    J  PIC 9 USAGE INDEX.
01    GRUPO-CALCULO.
  02    ELEMENTO-1 PIC X (5) VALUE SPACES.
  02    ELEMENTO-2 PIC A (4) VALUE SPACES.
  02    ELEMENTO-3 PIC 9 (3) VALUE ZEROS.
  .
  .
  .
```

El número de nivel 77 se coloca en el margen A, y todas las entradas de nivel 77 preceden a los otros niveles.

Obsérvese que a los elementos de GRUPO-CALCULO les hemos añadido la cláusula VALUE SPACES o VALUE ZEROS. No se confunda el que cuando se inicie el programa hayamos definido que estos elementos tengan ese valor, con que este sea su valor constante. Lejos de ello en el transcurso del programa estos elementos están destinados a contener muchos otros valores, son por lo tanto verdaderas variables, pero frecuentemente es conveniente que sepamos en el programa cuál es su valor inicial.

El formato de la sección de trabajo es:

```
DATA DIVISION.
FILE SECTION.
FD   (entradas de descripción de archivos y registros).
 .
 .
 .
WORK ING-STORAGE SECTION.
entr adas de nivel 77 si existen.
entr adas de descripción de registro si existen.
```

Si no se hace ninguna entrada, se suprime también el nombre de sección.

3.- Sección de constantes (CONSTANT SECTION)

Hemos de advertir que esta sección que pertenece a las normas CODASYL y USASI, y existente en los COBOL de muchos sistemas operativos, no existe en cambio en otros sistemas frecuentemente empleados.

Las entradas de esta sección son las mismas y en el mismo orden que las realizadas en la WORKING-SOTARAGE, un formato es por lo tanto:

```
DATA DIVISION.
FILE SECTION.
FD   (entradas de descripción de archivos y registros).
 .
 .
 .
WORK ING-STORAGE SECTION. (si existe)
     (entradas de nivel 77 y registro)
 .
 .
 .
CONS TANT SECTION.
77   (entradas de nivel 77 si las hay)
 .
 .
01   (entradas de registro si las hay)
 .
 .
```

La característica especial de las entradas en esta sección es que todas ellas tienen la cláusula VALUE, pero a diferencia de los valores iniciales dados en la WORKING-STORAGE, los de estos datos no se alteran durante la ejecución del programa. Como esto ocurriría lo mismo de haber sido definidos en la WORKING-STORAGE y el programador tuviese buen cuidado de no alterar sus valores (lo que de hecho se hace con los COBOL que no tienen esta sección) vamos a dar las razones que justifican su existencia.

Una.- Localización

El estar todas las constantes juntas favorece la documentación de los programas (tan importante) y el análisis. También favorece la facilidad de realizar cambios en el programa que afecten o no afecten a sus constantes.

Dos.- Compilación

El compilador al estar informado de qué datos deben ser constantes durante la ejecución del programa puede detectar durante la compilación errores como:

```
ADD  A  TO BASE
estando BASE definida en la CONSTANT SECTION
```

Un ejemplo que ilustrará el uso de constantes con nombres de datos es el siguiente:

En un programa de cálculo de nóminas, en el descuento de utilidades hay que tener en cuenta las deducciones autorizadas por Hacienda de modo que si el descuento es del 14 % y los ingresos son de 190.000 pesetas, con una deducción de 90.000, el descuento es el 14% de 100.000 pesetas o sea 14.000. Una fórmula para calcular esto sería:

```
COMPUTE DESCUENTO = (SUELDOS - DEDUCCION) * 0,14
```

y en la CONSTANT SECTION definiríamos

```
| 77 |  DEDUCCION PIC 9 (5) VALUE 90.000.
```

En la práctica hay más de un tipo de deducción, los cabezas de familia numerosa tienen otros importes, entonces sería muy conveniente en el programa tener nombres adecuados para cada uno de ellos. Por ejemplo:

DEDUCC-EMPLE, DEDUCC-FAM-1, DEDUCC-FAM-2

DEDUCC-OPER. que harían mucho más significativo el programa que la lectura de cantidades como 250 000, además de ser mucho más conveniente para cambiar estos tipos cuando así lo determina Hacienda.

TEMA X

TRANSFERENCIA DE LA INFORMACION

1.- El verbo MOVE

El verbo MOVE se utiliza para trasladar información de un área a otra, el área fuente no sufre alteración y la receptora se ve transformada para contener la información del área fuente.

El formato de este verbo es:

$$\underline{\text{MOVE}} \left\{ \begin{matrix} \text{nombre-datos-1} \\ \text{literal} \end{matrix} \right\} \underline{\text{To}} \text{ nombre-datos-2} \left[\text{, nombre-datos-3} \right] \ldots$$

Segunda opción:

$$\underline{\text{MOVE}} \left\{ \begin{matrix} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{matrix} \right\} \text{ nombre-datos-1 } \underline{\text{TO}} \text{ nombre-datos-2}$$

Los nombres - de - dato pueden ser de grupo o de elemento. Cuando se transmite a nivel de elemento se puede editar. También se puede editar con la segunda opción.

Las reglas generales de edición son:

1.- Datos numéricos

a.- Los datos del área fuente se alinean con el punto decimal en el área receptora. Cuando no existe el punto se alinean de derecha a izquierda.

Ejemplo:

```
     MOVE  A  TO  B
A PIC 999V99     si A valía 126.85 después de ejecutarse
B PIC  9V999     la declaración, B valdrá 6.85
```

b.- Si el PIC del área receptora contiene caracteres de edición: Z, -, +, etc., los datos del área fuente se editará de acuerdo con las reglas de edición explicadas en el Tema VIII.

Ejemplo:

```
     MOVE  A  TO  B
A PIC S99V99     si A valía -12,60 después de ejecutarse la
B PIC --9V99     declaración B valdrá -12.60
```

2.- Datos no numéricos

a.- El área fuente se traslada al área receptora ajustandose en ésta de izquierda a derecha.

b.- Cuando el área receptora es más grande, los espacios libres se rellenan de blancos.

c.- Cuando el área receptora es más pequeña se trunca a la derecha la transmisión. En este caso el compilador suele dar un mensaje de advertencia.

Ejemplo:

```
     MOVE  A  TO  B
A PIC A (6)      Si A contenía MADRID, B después de ejecutarse
B PIC A (5)      la declaración contendrá MADRI
```

Como guía de las transmisiones admisibles damos el siguiente cuadro, pero tengan en cuenta que estas posibilidades son siempre dependientes en la práctica al compilador con que se traduzca el programa existiendo importantes diferencias.

	Area receptora						
Area fuente	GR	AB	AN	N	ABE	ANE	NE
Grupo (GR)	S	S	S	N	S	S	N
Alfabético (AB)	S	S	S	N	S	S	N
Alfanumérico (AN)	S	S	S	N	N	S	N
Numérico (N)	S	N	S	S	N	S	S
Alfab. edit. (ABE)	S	S	S	N	S	S	N
Alfan. edit. (ANE)	S	N	S	N	N	S	N
Numérico edit. (NE)	S	N	S	N	N	S	N
ZEROS	S	N	S	S	N	S	S
SPACES	S	S	S	N	S	S	N

Con la segunda opción se transmiten sólo aquellas áreas (subdivisiones del área abarcada por el nombre de datos-1) que tienen el mismo nombre que áreas y a su vez divisiones del nombre de datos-2. También hay que tener en cuenta las calificaciones y éstas han de ser homogéneas. Por lo menos uno de los nombres-de-datos que se emparejan tiene que ser elemental. Cuando es preciso editar esto se ejecuta.

Con un ejemplo se ve mejor como funciona esta opción.

```
A          B
             03  GRUPO-1.
              04 ELE-1  PIC   9 (5).
              04 ELE-2.
              05 ELE-3 PIC A (6).
              05 ELE-4  PIC X (4).
             04  ELE-5 PIC  9(3)V99.
          02 GRUPO-2.
             05  ELE-1 PIC ZZ.ZZ9.
             05  ELE-2.
                 06  ELE-3 PIC X(7).
                 06  ELE-7 PIC X(3).
             05  ELE-8 PIC 9(3)V99.
```

la declaración

MOVE CORRESPONDING GRUPO-1 TO GRUPO-2

es equivalente a

MOVE ELE-1 OF GRUPO-1 TO ELE-1 OF GRUPO-2
MOVE ELE-3 OF ELE-3 IN GRUPO-1 TO ELE-3 OF ELE-2 IN GRUPO-2

2. - OPEN y CLOSE

Sabemos que tanto la información que entra en el ordenador como la que sale lo hace formando archivos, estos archivos están asociados a unidades físicas como son lectoras de fichas, de banda magnética, discos, perforadores, impresoras, etc. Pues bien antes de utilizar cualquiera de estas unidades para introducir o sacar información es preciso indicar en el programa que están dispuestos. Esta orden se da con el verbo OPEN cuyo formato es:

```
OPEN [ INPUT { nombre-archivo [ { REVERSED } ] } .... ]
     [                         [ { WITH NO REWIND } ]                ]
     [ OUTPUT { nombre-archivo  WITH NO REWIND } .... ]
```

Ejemplo:

OPEN INPUT ARCHIVO-MAESTRO, FICHERO OUTPUT IMPRESO

Como se ve en el formato no sólo se indican los nombres de los archivos sino también se especifica cuáles se emplean para introducir información y cuáles son de salida.

Cuando se trata de bandas u otros archivos que admiten etiquetas, al darse esta orden se leen y comprueban enviando mensajes cuando la etiqueta de un archivo input no concuerda con la que debía tener, o si es output, se va a grabar sobre un archivo cuya fecha de expiración no ha caducado todavía.

La opción REVERSED se emplea sólo con bandas que pueden leerse en sentido inverso.

La opción WITH NO REWIND se utiliza cuando en un carrete hay más de un archivo. Cuando se da la orden OPEN si la banda está avanzada se rebovina y es entonces cuando lee la etiqueta. Por lo tanto si quisiéramos leer un tercer o cuarto archivo, nos daría error. Para ello es necesario leer los archivos precedentes y cuando se abre este archivo se hace con la opción NO REWIND. Otro tanto ocurre cuando el archivo es output.

El verbo CLOSE realiza la función inversa pero ahora no se especifica qué archivos son input o output, esto ya es conocido por el compilador ya que para cerrar un archivo es preciso que previamente se haya abierto con un OPEN. Tanto los verbos OPEN como los CLOSE pueden aparecer desperdigados por el programa, sólo hay que respetar esta condición: que a cada OPEN de un determinado archivo le cierre posteriormente un CLOSE.

El formato del verbo CLOSE es:

CLOSE nombre-archivo [WITH {NO REWIND / LOCK}]
[, nombre-archivo [WITH {NO REWIND / LOCK}]] .....

la opción NO REWIND tiene la misma finalidad explicada en OPEN, la opción LOCK hace que el archivo así cerrado no puede ser abierto posteriormente durante este proceso.

Ejemplo:

```
CLOSE ARCHIVO-MAESTRO  WITH LOCK, FICHERO, IMPRESO
```

3.- READ

Para leer una ficha, un registro de banda o de otro medio de entrada se utiliza el verbo READ cuyo formato es: READ nombre-archivo RECORD [INTO nombre-dato] ; AT END declaración-imperativa ....

Previamente a dar una orden READ es necesario haber abierto el archivo con una orden OPEN y la opción INPUT, y no se puede dar

órdenes de lectura después de haberlo cerrado con una CLOSE. El nombre de archivo (el mismo empleado en el OPEN) debe estar definido por una entrada FD en la DATA DIVISION.

El resultado de esta operación es que se transmita el primer registro (dispuesto para leer. Ejemplo: la primera ficha), al área definida en la DATA DIVISION para contener los registros de este archivo. Aquí hay que recordar que si un archivo consta de dos otres tipos de registros, todos ellos comparten el mismo área y será labor del programador conocer cuál de los posibles registros es el que se ha leído.

Cuando el archivo en banda está organizado en bloques de más de un registro (digamos veinte) en realidad se leen estos veinte regis - tros de golpe en una área temporal (buffer) pero sólo uno de estos registros es transmitido al área definida en la DATA DIVISION para contener el registro o registros (si hay más de uno). En sucesivas órdenes READ se van leyendo uno a uno los otros hasta que se han leído los veinte, entonces es cuando se leen los otros veinte del siguiente bloque. Pero de toda esta operación no tiene que preocuparse el programador, estas funciones las crea automáticamente el compilador con la información que se le ha dado en la FD-entrada. Por lo tanto a todos los efectos podemos suponer que los registros se leen uno a uno.

Con la opción INTO, el registro se transmite también al área definida por el nombre-dato, como si después del READ se hubiera dado una orden MOVE. De esta forma la información queda en las dos áreas.

No se puede emplear la opción INTO cuando en el archivo hay más de un tipo de registro definido.

La condición AT END se hace cierta cuando se termina el archivo y al intentar leer un nuevo registro.

Después de haberse cumplido la declaración imperativa que sigue al AT END no puede intentarse una nueva lectura del archivo sin haber dado previamente una orden CLOSE y otra OPEN para este archivo.

Ejemplo:

READ ARCHIVO RECORD AT END GO TO CERRAR.

Cuando lo que se desea es sólo leer un dato desde una unidad fija del sistema como es la máquina de escribir de consola entonces se utiliza el verbo ACCEPT cuyo formato es:

ACCEPT nombre-dato [FROM nombre-memórico]

Ejemplo:

ACCEPT NUMERO FROM CONSOLE

CONSOLE puede ser el memórico dado en la ENVIRONMENT DIVISION como un SPECIAL-NAME. Otras veces es el único nombre que puede utilizarse, y en otros sistemas sólo es necesario escribir:

ACCEPT NUMERO

por ser único (la máquina de consola) el medio de entrada.

4. - WRITE

Con WRITE se cumplen funciones de salida de información tan diversas como escribir una línea de impresora, perforar una ficha o grabar un registro. Antes de poder dar una orden WRITE es necesario haber abierto el archivo de que se trata con un OPEN con la opción output. El formato de esta orden es:

WRITE nombre-registro [FROM nombre-dato-1]
[{BEFORE / AFTER} ADVANCING {nombre-dato-2 LINES / entero LINES / nombre-memórico}]

Obsérvese como aquí es nombre-registro y no nombre-de-archivo como en la orden READ. Esto es debido a que con WRITE hay que especificar cuál de los (posiblemente varios) registros del archivo queremos escribir.

Después de ejecutarse esta orden el contenido del área del registro queda destruida.

Para remediar esta circunstancia, si es necesario conservar la información, se utiliza la opción FROM. Entonces la información contenida en el área del nombre-dato-1 se traslada al área del registro como por un MOVE sin CORRESPONDING y se escribe seguidamente. El contenido del área registro se pierde, pero en cambio se conserva en nombre-dato-1.

Ejemplo:

Vamos a copiar un archivo en banda en otro

A	B
PROCEDURE DIVISION.	
P1.	OPEN INPUT ARCHIVO-V OUTPUT ARCHIVO-N
P2.	READ ARCHIVO-V AT END GO TO FINAL.
	WRITE REGISTRO-N FROM REGISTRO-V
	GO TO P2.
FINAL.	CLOSE ARCHIVO-V, ARCHIVO-N. STOP RUN.

La segunda opción se utiliza para perforar e imprimir. Lo vamos a comentar separadamente.

a.- Perforar.

Numerosas perforadoras tienen dos o más cajetines por donde pueden salir las fichas, entonces con una orden como

WRITE REGISTRO FROM DATO AFTER CAJA-1

haríamos que la ficha saliera por el cajetín primero de la perforadora. CAJA-1 será memórico definido en el párrafo SPECIAL-NAMES. Otras veces en un valor especial que puede tomar el nombre-dato-2 y la orden será así:

WRITE REGISTRO AFTER CAJA

Si CAJA vale A será el cajetín 1 y si es B será el cajetín 2.

b.- Imprimir.

Antes o después de imprimir es necesario indicar el número de líneas que debe avanzar la impresora para que se posicione en el lugar indicado. Esto se indica ya escribiendo

Ejemplo:

```
WRITE LINEA AFTER 5 LINES
WRITE LINEA BEFORE 3
WRITE LINEA AFTER I
```

En este último caso si I vale 2 al momento de ejecutarse la orden, la impresora avanzará dos líneas y entonces imprimirá. AFTER indica que el control de impresora se ejecuta antes de imprimir y BEFORE que se imprime y luego se ejecuta el control de impresora.

La condición de salto de página se puede indicar así:

```
WRITE LINEA FROM CABECERA AFTER SALTO
```

SALTO sería un memórico dado en el párrafo SPECIAL - NAMES.

En otros ordenadores esto se indica con un entero que tiene esta función (generalmente es el 0).

```
WRITE LINEA FROM CABECERA AFTER 0
```

Cuando sólo se desea escribir un breve mensaje en una unidad auxiliar fija en el sistema (generalmente la máquina de escribir de la consola) se utiliza el verbo DISPLAY cuyo formato es:

```
DISPLAY { literal-1      } [ , { literal-2      } ] ....
        { nombre-dato-1  }     { nombre-dato-2  }

[ UPON  nombre memórico ]
```

El nombre memórico suele ser CONSOLE y es fijo en el sistema o está definido en el párrafo SPECIAL-NAMES.

Ejemplo:

```
DISPLAY  167.50, 'PTS',  UPON CONSOLE
DISPLAY 'PRECIO ES', IMPORTE UPON CONSOLE
DISPLAY 'ERROR', NUM UPON CONSOLE
```

Un ejemplo del uso de la constante figurativa QUOTE es:

```
DISPLAY QUOTE 'SENTENCIA' QUOTE UPON CONSOLE
```

lo cual daría lugar que se escribiera

```
'SENTENCIA'
```

o sea que QUOTE nos permite escribir las comillas que en otro caso serían sólo marca del literal no numérico.

TEMA XI

DETERMINACIONES EXTERNAS

1. - División de identificación

La identificación del programa se detalla en la IDENTIFICATION DIVISION cuya organización y contenido es el siguiente:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. nombre-del-programa.
[AUTHOR [nombre] ... ]
[INSTALLATION. [nombre] ...]
[DATE-WRITTEN. [fecha ] ...]
[DATE-COMPILED. [fecha ] ...]
[SECURITY. [comentario ] ...]
[REMARKS. [comentario ] ...]
```

De todas las entradas la única indispensable es el nombre de la PROGRAM-ID. Según las normas CODASYL y USASI el nombre debe de ser una palabra que cumpla las reglas de formación de los nombres de datos. Sin embargo en importantes sistemas éste es un nombre externo (al COBOL) y debe de cumplir reglas especiales.

El resto de los comentarios (nombres, fechas, etc.) sólo es necesario que se escriban usando el juego de caracteres COBOL, dentro del margen B. La fecha de DATE-COMPILED la suele poner el ordenador cuando se realiza la compilación.

Ejemplo:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SA-451.
AUTHOR.  JUAN ESPAÑOL.
INSTALLATION.  CENTRO DE CALCULO.
DATE-WRITTEN.  31-VII-1969.
DATE-COMPILED.  28-VIII-1969.
SECURITY.  INFORMACION CONFIDENCIAL DIRECTOR
           GENERAL.
REMARKS.  IMPRIME LA SITUACION DE TRAMITACION Y
          EJECUCION DE LAS OBRAS DEL PLAN ASCE.
```

2.- División de contorno

En un programa COBOL se utilizan numerosos elementos físicos: lectoras de fichas, perforadoras, impresoras, discos, bandas magnéticas, etc. Es frecuente el uso de media docena de armarios de bandas, de dos impresoras, y así de otras unidades iguales que es necesario identificar. En el programa es sumamente simple pues se les da nombres llamados lógicos, pero hay que asociarlos a las unidades físicas.

Además de este problema hay que tener en cuenta que una de las frialidades del COBOL como lenguaje es que los programas sean independientes de la instalación donde han de procesarse.

Este ideal, al parecer imposible de alcanzar en la actualidad, se reduce en COBOL a especificar las características especiales de la instalación en la ENVIRONMENT DIVISION con lo que el problema queda simplificado y localizado.

(NOTA: Lamentablemente esto no ocurre plenamente. Hay importantes sistemas que se diferencian de otros en la DATA DIVISION, principalmente en la entrada de descripción de archivos (FD), en los nombres COMPUTATIONAL, en la CONSTANT SECTION y en el enlace de áreas comunes con subrùtinas. En la PROCEDURE también hay diferencias en las declarativas, en los verbos usados y en los métodos de controlar archivos y el carro de impresora.)

En la ENVIRONMENT DIVISION se relacionan los nombres lógicos que se dan en el programa a los elementos físicos, con los nombres físicos que en una determinada instalación reciben las unidades que

los contienen. Así se relaciona el nombre de un archivo que puede ser un informe impreso con el nombre que en la instalación se da a la impresora con la que se va a imprimir.

También se puede asociar nombres memóricos con funciones especiales como salto de página, o situaciones de un SWITCH o con nombres de unidades como la máquina de escribir de consola.

La organización de esta división es:

```
A   B
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.....
OBJECT-COMPUTER.....
SPECIAL-NAMES ....
INPUT-OUTPUT SECTION
FILE-CONTROL ....
I-O-CONTROL ....
```

3. - Sección de configuración

La sección de configuración detalla las especificaciones del sistema de proceso de datos donde se va a operar el programa. Comprende tres párrafos cuyos nombres son: SOURCE-COMPUTER, OBJECT-COMPUTER y SPECIAL-NAMES.

SOURCE -COMPUTER tiene el formato:

```
SOURCE- COMPUTER. nombre del ordenador.
```

OBJECT- COMPUTER tiene el formato:

```
OBJECT- COMPUTER. nombre del ordenador.
        [MEMORY SIZE entero {WORDS     }]
                            {CHARACTERS}
                            {MODULES   }
```

Ejemplo:

```
SOURCE-COMPUTER. NAME-833
OBJECT-COMPUTER. NAME-825 MEMORY 20000 WORDS.
```

SPECIAL-NAMES tiene el formato:

```
SPECIAL NAMES. nombre-físico { IS memórico-1 [ ON STATUS IS ]
                             { IS memórico-2 [ OFF STATUS IS ]
                             { ON STATUS IS nombre-condición-5
                             { OFF STATUS IS nombre-condición-7

nombre-condición-1, OFF STATUS IS nombre-condición-2 }
nombre-condición-3, ON STATUS IS nombre-condición-4  }
, OFF STATUS IS nombre-condición-6                   }
, ON STATUS IS nombre-condición-8                    }

    [ DECIMAL-POINT IS COMMA ]
```

Ejemplo:

```
SPECIAL-NAMES.
    FORM-OVERFLØW ON IS FIN-PAGINA OFF IS SIGUE,
    DECIMAL-POINT ES COMMA.
```

DECIMAL-POINT IS COMMA indica que se cambie el punto por la coma en todas las funciones de edición definidas por las cláusulas PIC.

A los SWITCH se les puede dar nombre memórico y nombres de condición.

Se pueden dar nombres memóricos también a unidades de hardware que se usan con los verbos DISPLAY y ACCEPT.

4. - Sección de entrada y salida .

Esta sección llamada INPUT-OUTPUT SECTION se compone de dos párrafos: FILE-CONTROL que relaciona los archivos utilizados por el programa con las unidades físicas empleadas y I-O-CONTROL que define técnicas especiales de entrada y salida.

```
FILE CONTROL { SELECT [OPTIONAL] nombre-archivo
ASSIGN TO [entero-1] nombre-físico, [nombre-físico-2] ...
[FOR MULTIPLE REEL ], RESERVE { entero-2 } ALTERNATE [ { AREA  } ]
                              { NO       }            { AREAS }
                                                                  }...
```

OPTIONAL se utiliza para archivos que pueden estar o no presentes cuando el programa es ejecutado.

Los nombres de archivo que aparecen en este párrafo tienen que estar definidos por una entrada FD en la FILE SECTION de la DATA DIVISION.

Entero-1 indica el número de unidades de entrada y salida que se asignan al archivo, de no indicarlo el compilador es el que lo determina.

El nombre-físico es nombre dado en cada instalación partcular. Este es uno de los particulares que son dependientes de cada máquina y que el programador debe cumplimentar consultando el manual correspondiente. Este nombre físico puede ser un armario de bandas, un lector de fichas, una impresora, etc.

Si se trata de un archivo en banda magnética y éste cubre más de un carrete, entonces se especifica FOR MULTIPLE REEL.

La lectura o escritura de registros puede ser muy acelerada si se dispone de un área alternativa de forma que las operaciones de entrada y salida puedan solaparse unas con otras y con el cálculo en la unidad central. Entero-2 indica el número de áreas alternativas que se reservan y por lo tanto cuando se especifica NO, no se reserva ningún área alternativa. Esto es otra de las opciones que no se pueden rellenar acertadamente sin conocer bien la instalación donde se ve a ejecutar el proceso.

```
I-O-CONTROL  [ RERUN [ ON { nombre-archivo-1 } ] ]
                          { nombre-sistema   }

  EVERY { END OF REEL                                  } { OF nombre-archivo-2 } ]
        { entero-1 RECORDS                             }
        { entero-2 CLOCK-UNITS                         }
        { nombre-condicional                           }

[; SAME REDORD AREA FOR nombre-archivo-3, nombre-archivo-4
                                                  .... ] ...

[; MULTIPLE FILE TAPE CONTAINS nombre-archivo-5 [ POSITION
                                                   entero-
[; nombre-archivo-6 [ POSITION entero-4 ] ] ... ] ....
```

La declaración RERUN indica puntos de control. Se emplea en procesos de muy larga duración. En estos casos un fallo mecánico o simplemente una falta en el servicio eléctrico puede dar por resultado la pérdida de todo el trabajo realizado hasta entonces.

Con la declaración RERUN se hace almacenar en una memoria exterior (banda magnética o disco), el contenido de la memoria en el momento del control, de forma que pueda reanudarse el trabajo a partir de este momento.

El estudio de cómo organizar estos puntos de control y la forma de reanudar un trabajo desde el lugar interrumpido es una técnica avanzada que queda fuera de esta introducción.

Señalaremos sólo que:

nombre-archivo indica el archivo (generalmente banda) donde se hace el almacenamiento. El lugar donde se hace está determinado por cada instalación.

nombre-sistema indica la unidad física única donde se hacen todos los almacenamientos.

EVERY END OF REEL OF nombre-archivo-2 indica el momento (final de cada carrete) en que se hace el almacenamiento.

EVERY entero-1 RECORDS OF nombre-archivo-2 indica el número de registros procesados del archivo, al final de los cuales se hace un almacenamiento.

EVERY entero-2 CLOCK-UNITS indica el intervalo de tiempo al final del cual se hace el almacenamiento.

nombre de condición es el de un SWITCH que se ha especificado en el párrafo SPECIAL-NAMES.

$$\left[\text{SAME}\left[\text{RECORD}\right]\right]\ \text{AREA FOR nombre-archivo-3}\left\{,\ \text{nombre-archivo-4}\right]\ldots$$

...

Cuando es necesario economizar memoria se puede compartir el área de entrada y salida entre varios archivos. Hay que tener en cuenta entonces en que si se especifica RECORD sólo un registro de to

dos los archivos especificados puede estar presente en un momento especificado.

[MULTIPLE FILE TAPE CONTAINS nombre-archivo-5
[POSITION entero-3] [, nombre-archivo-6 [POSITION entero-4]]...]..

se utiliza cuando en un carrete hay más de un archivo. Si estos archivos están colocados en secuencia no es necesario indicar POSITION.