

PROGRAMACIÓN EN COBOL

Desarrollar aplicaciones informáticas en lenguaje de programación COBOL.

50 horas

Estructura del lenguaje COBOL.

COBOL es un lenguaje de alto nivel y de fácil entendimiento basado en la resolución de problemas de los campos de gestión y documentación. COBOL son las iniciales de COMmon BUssiness ORiented Language.

Aunque hay lenguajes que nos permiten realizar programas mucho más complejos en cuanto a posibilidades que aquellos que podemos realizar con COBOL, tenemos que decir que este lenguaje de programación por el momento no solo que no va a desaparecer sino que va a perdurar mucho más tiempo que mucho de los posteriores, ya que hay muchas empresas que siguen usando COBOL desde el principio de estas, y como sus programas funcionan constantemente, no se pueden apagar para hacer el cambio a otro lenguaje además del gran coste que supondría el tomar esta opción.

Por otra parte hay que destacar que el COBOL no nos permite programar juegos complicados para el ordenador, sin embargo si queremos hacer una programación de bases de datos, tendremos que tener en cuenta que es mucho más fácil para nosotros el realizarla con COBOL, que el tener que programarla desde cero con programas como JAVA o C++.

La estructura de COBOL es totalmente diferente a la estructura de cualquier otro lenguaje. Esta especialización impide aunque de forma muy simple que expertos programadores vean COBOL como diferente e inaccesible para ellos debido a las grandes diferencias que se encuentran con la programación tradicional estructurada. Sin embargo aprender COBOL es mucho más fácil que aprender los otros lenguajes que existen en el mercado, con el inconveniente de no poder hacer de todo con él.

Un inconveniente que tiene COBOL a la hora de programar es lo estricto que es cuando se definen las variables, sin embargo, gracias a esta precisión hace que el programa resulte más rápido que si estuviera realizado por otros lenguajes.

Juego de caracteres.

COBOL permite utilizar casi todos los caracteres que permite el teclado, además de distinguir entre mayúsculas y minúsculas. Un carácter a mencionar por su utilidad y porque en otros lenguajes no es usado sería el espacio.

Hay tres tipos de juegos de caracteres, los alfabéticos, los numéricos y los especiales.

Los alfabéticos corresponden a los 26 caracteres usados en todas las lenguas europeas, usándose tanto en mayúsculas como en minúsculas. Como carácter alfabético también incluiremos el espacio.

Los numéricos corresponden a los diez dígitos con los que podemos representar cualquier número.

Los caracteres especiales que son los siguientes:

+	Suma	.	Punto
-	Resta	"	Comillas
*	Asterisco	(Paréntesis Izquierdo
/	Barra de división)	Paréntesis Derecho
=	Igual	>	Signo de Mayor que
\$	Dolar	<	Signo de Menor que
,	Coma	:	Dos Puntos
;	Punto y Coma	&	Ampersand

Palabras en COBOL:

Hay cuatro tipos de palabras en COBOL. Las palabras definidas para los usuarios, los nombres de funciones, palabras reservadas y los nombres del sistema.

Ninguna de las palabras puede en ningún caso superar los 30 caracteres de longitud. Deben formarse por caracteres numéricos o alfabéticos, incluyéndose el guión (signo menos) salvo al principio de la palabra.

Hay que destacar que las palabras en minúsculas son equivalentes a las que están formadas por mayúsculas, por lo que no se podrán utilizar ambas como distintas.

Palabras reservadas de COBOL, variables, símbolos y constantes.

El punto es un signo de vital importancia en cobol ya que nos indica el final de una línea, en el han de terminar todas las secciones, divisiones y párrafos. Si al final de una línea el compilador no encuentra el punto, interpretará que la instrucción continúa hasta que aparezca el punto de fin de línea.

Dentro de COBOL hay una serie de palabras que están reservadas al lenguaje y que por ese motivo no se pueden utilizar para definir campos o variables.

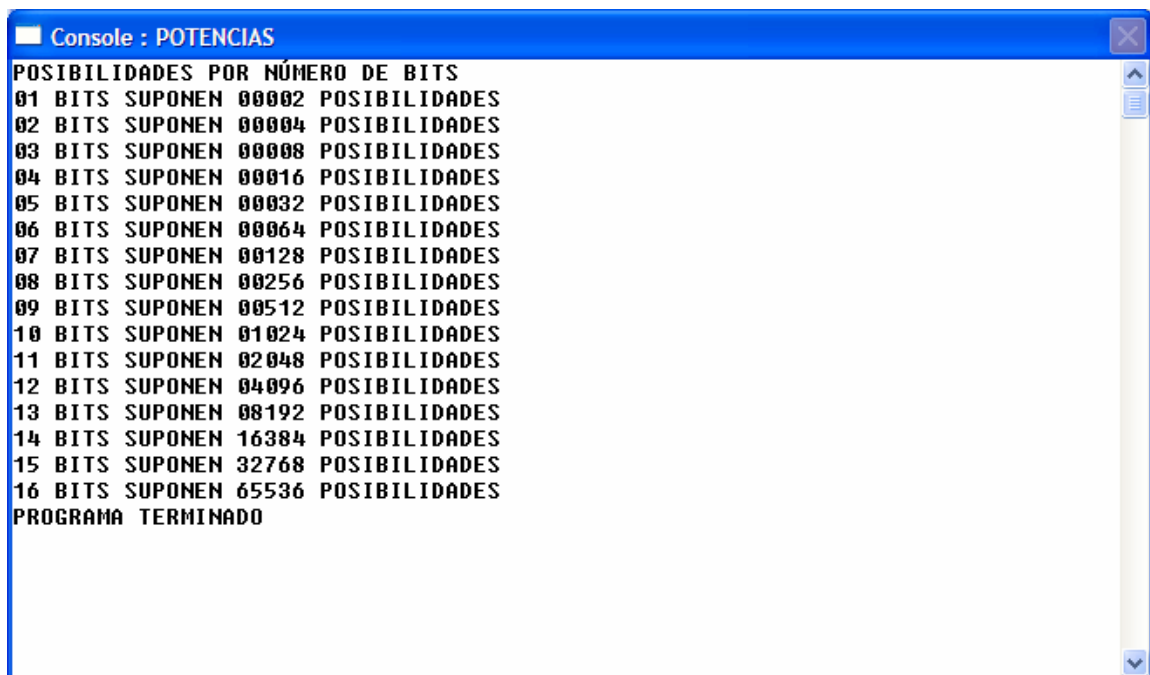
Las variables sin embargo son creadas por el programador, este tipo de palabras admiten hasta treinta caracteres (letras, dígitos o guiones) que comenzarán siempre por una letra.

Los símbolos son aquellos que tienen un significado específico para COBOL como tales, y pueden ser ortográficos(., ;), aritméticos(+, -, *, /, **) y relacionales (<, >, =).

Las constantes son los valores fijos utilizados a lo largo del programa. Tanto el nombre como el valor vienen programados por el programador. Las constantes pueden ser numéricas (su valor es un número de hasta 18 dígitos.), no numéricas (vienen representadas entre comillas) y las figurativas (son palabras reservadas que tienen un valor constante (ZERO = 0, SPACE = " ", HIGH-VALUE = El valor más alto, LOW-VALUE = El valor más bajo).

Ejercicio de Ejemplo:

El siguiente programa se encarga de mostrar por pantalla las diferentes posibilidades que se pueden dar con un número de bits que van de 1 a 16 (de 1 bit a 2 bytes que son la unidad física mínima y la unidad lógica mínima que se utilizan actualmente en informática).



```
Console : POTENCIAS
POSIBILIDADES POR NÚMERO DE BITS
01 BITS SUPONEN 00002 POSIBILIDADES
02 BITS SUPONEN 00004 POSIBILIDADES
03 BITS SUPONEN 00008 POSIBILIDADES
04 BITS SUPONEN 00016 POSIBILIDADES
05 BITS SUPONEN 00032 POSIBILIDADES
06 BITS SUPONEN 00064 POSIBILIDADES
07 BITS SUPONEN 00128 POSIBILIDADES
08 BITS SUPONEN 00256 POSIBILIDADES
09 BITS SUPONEN 00512 POSIBILIDADES
10 BITS SUPONEN 01024 POSIBILIDADES
11 BITS SUPONEN 02048 POSIBILIDADES
12 BITS SUPONEN 04096 POSIBILIDADES
13 BITS SUPONEN 08192 POSIBILIDADES
14 BITS SUPONEN 16384 POSIBILIDADES
15 BITS SUPONEN 32768 POSIBILIDADES
16 BITS SUPONEN 65536 POSIBILIDADES
PROGRAMA TERMINADO
```

Introducir el siguiente programa en el ordenador para ver el resultado.

IDENTIFICATION DIVISION.

PROGRAM-ID. POTENCIAS.

AUTHOR. jlbv@jlbv.com.

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 AUX.

02 X PIC 9.
02 N PIC 9.
02 MAX PIC 9.
02 POT PIC 9(5).

PROCEDURE DIVISION.

MD1-INICIO

MOVE 2 TO X.
MOVE 1 TO N.
MOVE 16 TO MAX.
DISPLAY 'POSIBILIDADES POR NÚMERO DE BITS'.

MD2-TRATAMIENTO

PERFORM MD3-RUTINA THRU FIN-MD3 UNTIL N > MAX
DISPLAY 'PROGRAMA TERMINADO'
STOP RUN.

MD3-RUTINA

COMPUTE POT = X ** N.
DISPLAY N ' BITS SUPONEN ' POT ' POSIBILIDADES'.
ADD 1 TO N

FIN-MD3.

EXIT

Estructura de los programas COBOL.

Organización de los datos.

Todos los números llevan un valor que llamado número de nivel que indica la jerarquía de los mismos en la estructura general de los datos.

Los números de nivel van del 01 hasta el 49, aunque también existen el 66, 77 y 88. El 01 define al campo de más importancia, y que viene seguido del 02 que tiene menos importancia.

Declaraciones, sentencias, párrafos, y divisiones.

Las declaraciones son conjuntos de dos o más palabras reservadas o palabras que especifican una operación (generalmente comienzan por un verbo).

Una sentencia es el conjunto de una o más declaraciones, quedando definida la última de ellas porque termina con un punto.

El párrafo es por tanto un conjunto de sentencias que vienen precedidas por un título.

Por último tendremos que destacar la división, que surgen por las cuatro divisiones naturales de la programación en COBOL que son:

- IDENTIFICATION DIVISION
- ENVIRONMENT DIVISION
- DATA DIVISION
- PROCEDURE DIVISION

Programa Fuente.

A la hora de realizar un programa fuente, lo deberemos realizar siguiendo las normas del compilador de COBOL. Las normas más estandarizadas son las siguientes:

Todas las líneas tienen 80 caracteres, ya que en MS-DOS la pantalla sólo tenía estas columnas, y dado que muchas versiones de COBOL funcionan en MS-DOS, se intenta seguir esa pauta, aunque en Windows aparezcan más de 80 caracteres y además con la barra de desplazamiento horizontal podamos obtener muchos más.

El programa cobol se escribe secuencialmente en líneas de 80 caracteres o menos con la siguiente división:

```

1 2 3 4 5 6 7 8
-----
000001*AAAA

```

La parte (1) comprende las columnas de la 1 a la 6 ambas inclusive y se utiliza para numerar las líneas, aunque hoy en día prácticamente no se utilizan, ya que se suele hacer de forma automática.

La parte (2) comprende la columna 7 y es la que habla con el compilador directamente. En ella podemos encontrar, un guión (-) que nos indica que esta línea es continuación de la anterior pero que por su tamaño ocupa mas de una línea, un asterisco (*) que nos indica que el texto que viene a continuación es un comentario y por lo tanto que el compilador lo ignore, una barra (/) indica que se debe saltar una página, o bien puede servir dependiendo de los compiladores para indicaciones del debug.

La parte (3) se le llama área A comprende las columnas 8 a 11 ambas inclusive y aquí es donde se escriben los nombre de las divisiones, de las secciones, de los párrafos, los indicadores de FD (File Description) y los niveles de variables 01 y 77.

La parte (4) llamada área B comprende desde la columna 12 a la 72 y en ellas se incluirán todas las instrucciones del programa, las líneas de las secciones y los niveles de variables mayores a 01.

La parte (5) de la columna 73 a la 80 no se utiliza y por lo tanto es ignorada por el compilador.

Hay instrucciones que sólo pueden funcionar en las líneas 8 a 11 (Zona A), otras que solo pueden funcionar en las líneas 12 a 72 (Zona B), y por último también hay otras que pueden funcionar en ambas zonas indistintamente.

Los comentarios se representarán mediante líneas que empiecen con * o /, y se podrán escribir tanto en la zona A como en la B.

Si una frase no cabe en una línea, se podrá utilizar el símbolo – al principio de la siguiente línea en la zona A para indicar que ahí continúa la línea anterior.

Divisiones de un programa COBOL.

Un programa de COBOL tiene cuatro divisiones. Todos los programas deben tener las cuatro secciones, aunque alguna de ellas esté vacía.

Todas las divisiones deben comenzar en la zona A de programación (de las columnas 8 a 11).

IDENTIFICATION DIVISION.

Esta división se encarga de identificar el programa. También es la parte menos importante, sobretodo en el caso de ser sólo un programa.

PROGRAM ID.

Es el único párrafo obligatorio dentro de la IDENTIFICATION DIVISION y en él se identifica el nombre del programa. Debido a su compatibilidad con el DOS, el nombre de del archivo no podrá tener nunca más de 8 elementos. Si por algún motivo se intentara guardar el archivo con más de 8 caracteres el ordenador sólo necesitará los 8 primeros para llamar al archivo.

AUTHOR.

Es una división opcional que permite escribir el nombre del autor del programa.

INSTALLATION.

En esta división también opcional se suelen definir tres elementos, el nombre del que realiza la instalación, el equipo que a probado o testado la instalación o las características necesarias del ordenador para poder instalar el programa.

DATE-WRITTEN.

Indica la fecha en la cual se realizó el código fuente del programa. Gracias a esta anotación, si encontramos dos programas que parecen iguales pero con fechas diferentes, supondremos que el que tiene la última actualización será el que tiene la fecha posterior.

DATE-COMPILED.

Esta división nos permite ver la fecha en la cual se compiló el programa. Por lo general esta fecha es posterior o igual a la del programa, aunque en el caso de que se esté mejorando el programa, esta fecha significa que ese programa todavía no ha conseguido una versión estable desde la que se hizo en el momento de la compilación.

REMARKS.

Sirve para incluir cualquier otro comentario hasta aquí no reflejado.

Vemos que el único párrafo obligatorio además del nombre de división es el que hace referencia al nombre del programa, los demás nombre de autor, lugar de instalación, fechas de creación y compilación y comentarios son opcionales, eso sí, si se incluyen se deben de poner cumpliendo las normas.

ENVIRONMENT DIVISION.

Es la segunda división por orden de aparición, y en ella se especifican, el ordenador donde se escribió y se ejecutará el programa, así como la relación entre los ficheros a utilizar con sus correspondencias externas, es decir con los dispositivos a los que hará referencia el programa objeto cuando vaya a establecer comunicación con dicho fichero.

Diremos antes de continuar, que en las primeras versiones de este lenguaje había muchas partes que eran obligatorias en cada programa, pero hoy en día, por ejemplo, ésta división ya no es obligatoria, así como ninguna de sus partes.

Esta división es obligatoria aunque no se manejen ficheros en el programa. Incluye nombres de sección y párrafos fijos que pueden omitirse si no son necesarios.

Aquí es donde indicamos al programa de donde se deben tomar los datos y que salidas se utilizarán.

Esta división empieza en la columna 8 y los nombres de párrafo y sección son fijos.

Esta división se divide a su vez en dos partes claramente diferenciadas. La primera, la CONFIGURATION SECTION nos permite especificar las diferentes características del equipo sobre el que se escribió el programa y en el cual debe funcionar, o bien el nombre del compilador y asignación de valores a ciertas constantes utilizadas por el compilador. La segunda, la INPUT-OUTPUT SECTION nos permite especificar los diferentes ficheros con los que trabajará el programa además de su organización, método de acceso, nombre de fichero, etc.

CONFIGURATION SECTION.

SOURCE-COMPUTER.

Para especificar el tipo de ordenador servidor del programa.

OBJECT- COMPUTER.

Son las características del ordenador remoto que accede al programa.

ESPECIAL NAMES.

En él se especifican entre otros aspectos el tipo de símbolo para separar los decimales, signo monetario, etc.

Al haber varias opciones diferentes, tan sólo en la última debe aparecer el punto.

Para la línea de SPECIAL-NAMES el uso más habitual es el de cambiar el punto decimal usado por los ingleses por la coma y así poder especificar los puntos para los miles, su formato sería el siguiente:

SPECIAL-NAMES.

DECIMAL-POINT IS COMMA.

También podríamos cambiar el valor del símbolo de la moneda con:

CURRENCY SIGN IS literal, suele ser un solo carácter y no puede coincidir con ninguno de los que usamos para definir las variables, es decir ni A, ni Z, ni 9, ni -, ni +, ni X, etcétera.

O hacer que todas las letras introducidas sean mayúsculas o minúsculas o que no haya diferencias entre ambas con la cláusula ALPHABET.

INPUT-OUTPUT SECTION.

Las instrucciones que se utilizan en esta sección son los siguientes:

FILE-CONTROL.

SELECT [OPTIONAL] Nombre-de-archivo

ASSIGN TO Tipo-de-dispositivo

ORGANIZATION IS Tipo de organización

ACCESS MODE IS Modo de acceso al fichero

RECORD KEY IS Clave del registro.

ALTERNATE RECORD KEY IS Claves alternativas al registro.

WITH DUPLICATES

FILE STATUS IS Variable de estado del fichero.

FILE CONTROL.

SELECT.

Es aquí donde especificamos el nombre lógico que va a tener el fichero dentro del programa, suele ser una palabra que identifique lo mas claro posible el contenido del fichero, por ejemplo ARTICULOS, PROVEEDORES, CLIENTES.

Si indicamos la opción OPTIONAL, al hacer un OPEN I-O, si el archivo no existe, se crea. Con lo cual nos evitamos tener que abrirlo como OUTPUT y cerrarlo, antes de poder utilizarlo por primera vez.

ASSIGN TO.

Con esta cláusula, especificamos el tipo de dispositivo, si es una impresora PRINTER, si es un fichero sobre el que vamos a grabar RANDOM o DISC, se pueden utilizar otros como INPUT, INPUT-OUTPUT, CASSETTE, MAGNETIC-TAPE, pero sin duda los más utilizados son los dos primeros para identificar si el fichero utilizará una salida impresa o se utilizará sobre disco. Para identificar ficheros utilizados para clasificar utilizaremos SORT.

ORGANIZATION IS.

Aquí indicamos la organización de los registros de nuestro fichero, podrá ser SEQUENTIAL, RELATIVE o INDEXED, si nuestro archivo fuera secuencial se podría omitir ésta cláusula así como las restantes.

De ésta organización se deriva el formato del fichero, SEQUENTIAL si los registros se graban secuencialmente conforme se dan entrada sin importar si están o no repetidos, un ejemplo claro son los archivos de impresora, todos los listados son secuenciales.

RELATIVE, si cada registro es identificado por un valor entero con su posición relativa (prácticamente no se utiliza).

INDEXED es la mas utilizada e identifica a ficheros que sus registros son accesibles mediante una clave única e irrepetible o por varias que pueden estar duplicadas, cualquier fichero de mantenimiento, por ejemplo de ARTICULOS, podría ser INDEXED, y cada código será único para cada artículo y con el nos iremos a su posición y podremos ver todos los demás datos que hagan referencia al registro. Existe también para los archivos de texto, tipo AUTOEXEC.BAT la posibilidad de asignarlos directamente especificando LINE SEQUENTIAL en ésta cláusula.

ACCESS MODE IS.

Indica el modo de acceso al fichero, puede ser SEQUENTIAL, RANDOM o DYNAMIC, si no se especifica ninguno o si el fichero es SEQUENTIAL entiende que el modo será SEQUENTIAL.

RANDOM indica que accederemos a él de forma aleatoria por su clave y DYNAMIC (la mas utilizada) con la que podremos acceder al fichero en el modo que queramos dentro del programa, unas veces de forma secuencial, si nos interesa, otras veces por su clave.

RECORD KEY.

Se utiliza sólo si el fichero es indexado y en el decimos cual es el nombre de la clave por la cual accederemos a los registros. Esta deberá ser alfanumérica y tendrá que estar especificada en la FD del fichero. Si el archivo fuera RELATIVE, esta cláusula se sustituiría por RELATIVE KEY e indicará el número de

registro del fichero, deberá estar declarado en la WORKING-STORAGE SECTION como una variable numérica sin signo.

ALTERNATE RECORD KEY.

Esta cláusula se utiliza solo para ficheros indexados e identifican una o mas claves alternadas para nuestros registros, por ejemplo en un fichero de clientes cuya clave principal sería el código, podríamos asignar como clave alternativa el NIF, y podríamos acceder a el por las dos claves, bien por código o bien por NIF, será también alfanumérico y deberá también estar declarado en la FD. Si aparece WITH DUPLICATES, indica que ésta clave alternativa pudiera estar duplicada, por ejemplo si hubiéramos escogido como clave alternada además del NIF, el Nombre del cliente, podría darse el caso de que dos clientes tuvieran el mismo nombre.

FILE STATUS.

Aquí damos el nombre de una variable que especificaremos en la WORKING STORAGE SECTION como un campo alfanumérico de dos caracteres donde el programa depositará el código de error que ocurra en el fichero, dependiendo del valor nosotros podremos operar o hacer alguna acción en concreto.

I-O CONTROL.

Se utiliza para indicarle al programa cuantos archivos van a utilizar el mismo área de memoria para trabajar. Actualmente no se utiliza mucho debido a la gran potencia de los ordenadores.

Un ejemplo de una ENVIRONMENT DIVISION, podría ser la siguiente:

CONFIGURATION SECTION.

SOURCE-COMPUTER. FSCOBOL

OBJECT-COMPUTER. FSCOBOL

SPECIAL-NAMES. DECIMAL-POINT IS COMMA.

INPUT-OUTPUT SECTION.

FILE-CONTROL

SELECT CLIENTES ASSIGN TO RANDOM "C:DATOSCLIENTES.DAT" ORGANIZATION INDEXED
ACCESS DYNAMIC

RECORD KEY KEY-CLIENTE

ALTERNATE RECORD KEY-CLIENTE1

FILE STATUS STACLI.

SELECT IMPRESORA ASSIGN TO PRINT "PRINTER

DATA DIVISION.

Es la tercera división por orden de aparición, y es donde se declaran absolutamente todos los nombres de campos, registros, variables, es decir donde nombramos cada dato que vayamos a utilizar en nuestro programa. Para almacenar todos estos nombres de datos, ésta DIVISION se divide en varias secciones, cada una de ellas orientada a un tipo de datos diferente.

WORKING-STORAGE SECTION.

Al igual que en la anterior DIVISION ninguna de sus partes es obligatoria, pero si vamos a utilizar alguna variable, aunque solo sea una, tendremos que incluirla en la WORKING-STORAGE SECTION y esto nos obligará también a definir la DATA DIVISION. Pasemos a continuación a explicar más a fondo cada una de ellas.

Esta penúltima división nos permite definir todos los campos de datos a utilizar. También encontraremos una sección para transmisión de datos a otros programas.

Todos los párrafos de esta división son opcionales, aunque en algunos compiladores estos párrafos son obligatorios. Al igual que en la anterior DIVISION ninguna de sus partes es obligatoria, pero si vamos a utilizar alguna variable, aunque solo sea una, tendremos que incluirla en la WORKING-STORAGE SECTION y esto nos obligará también a definir la DATA DIVISION.

Toda la DATA DIVISION se programa en la zona B.

FILE SECTION.

Aquí describiremos todos los campos que componen los registros de todos los archivos que vayamos a utilizar, que previamente habremos declarado en la INPUT-OUTPUT SECTION dentro de la ENVIRONMENT DIVISION.

El resumen de una FILE SECTION podría ser la siguiente:

FD Nombre del fichero .

BLOCK CONTAINS Número de registros por bloque **RECORDS**

RECORD CONTAINS Número de caracteres por registro **CHARACTERS**

LABEL RECORD Etiqueta de registro

DATA RECORD Nombre del registro.

Para identificar el nombre del fichero utilizaremos la cláusula FD que previamente habíamos descrito en la cláusula SELECT de la INPUT-OUTPUT SECTION en la ENVIRONMENT DIVISION.

Cuando queremos que por cada bloque en disco se graben mas de un registro usaremos la cláusula BLOCK CONTAINS, en ella especificamos el número de registros que pueden caber en cada bloque, (512, 1024), si no se especifica se supone que cada registro va a ocupar un bloque de memoria, o bien será el propio compilador el que haga el cálculo mas apropiado.

Para saber el número de caracteres que queremos tener en cada registro, usaremos RECORD CONTAINS, que define el número de caracteres que tiene el registro sumando todos sus campos, puede ser fija o variable. Si es fija utilizamos un valor y si es variable un rango desde hasta, si no se especifica será el propio compilador quien la determine.

La cláusula LABEL RECORD puede tener dos valores STANDARD u OMITTED, el primer caso indica que cada vez que se accede a un registro el compilador hará las comprobaciones estandares descritas por el propio compilador y en el segundo éstas serán omitidas. Para el caso de los ficheros de datos en disco se suele poner STANDARD y cuando el fichero es de impresora se indica OMITTED.

Por último, en DATA RECORD debido a que un mismo fichero puede tener varias descripciones de registro, indicáramos los nombres de éstas que deberán estar descritas a nivel 01. Normalmente no se utiliza y casi siempre se utilizan una sola descripción por fichero, por lo que no suele aparecer en casi ningún programa.

WORKING STORAGE SECTION.

Los campos de trabajo que se van creando para el programa.

Todos los campos llevarán su número de nivel en la zona B, a excepción de los niveles 01 y 77 que se definirán en la zona A.

En ella declararemos todas las variables no referentes a archivos, pero que durante la ejecución del programa vayamos a utilizar.

LINKAGE SECTION.

Esta es la sección donde se registrarán las variables que nos servirán para enlazar el programa principal con el que llamemos mediante la orden CALL.

Tanto los datos introducidos en la WORKING STORAGE SECTION como los que se encuentra en la LINKAGE SECTION tienen la misma estructura: Número de Nivel Nombre de campo PIC, VALUE, REDEFINES, OCCURS, JUST, SIGN, SYNC.

Cada campo declarado debe de llevar un número de nivel que le informe al compilador del tipo de campo que es:

El nivel 01, identifica la primera entrada de un registro o la primera entrada de un campo que se va a subdividir.

El nivel 77, identifica a una variable que no se va a subdividir y que no forma parte de ningún registro.

El nivel 88, identifica los posibles valores condicionales de una variable previamente definida.

Los niveles 02 al 49 indicarán las distintas subdivisiones de un campo cuya primera entrada ha sido definida a nivel 01. Los niveles 01 y 77 deberán de ir siempre en el Area A (Col 8) el resto es independiente.

A continuación pondremos el nombre del campo, que no podrá ser ninguna palabra cobol ni llevar ningún carácter extraño, principalmente se utilizarán letras y números o guiones. Es posible que algún campo que definamos nunca vaya a ser usado por el programa pero si en cambio es necesario que exista para que nos reserve el espacio, le llamaremos FILLER.

Y finalmente podrán venir cláusulas como:

PICTURE / PIC esta palabra es la que utilizamos para identificar el tipo de datos que va a contener la variable. Los posibles valores son:

DE CAMPOS.

9 - Para campos numéricos.

A - Para campos alfabéticos.

X - Para campos alfanuméricos.

S - Indica variable con signo.

V - Indica punto decimal.

DE EDICIÓN.

\$ - Representa la aparición del signo \$ delante del campo numérico.

. - Indica separación de miles.

, - Indica punto decimal. (estas dos pueden variar según hayamos especificado en SPECIAL-NAMES DECIMAL-POINT IS COMMA).

Z - Representa un espacio para el 0 a la izquierda en campos numéricos.

* - Igual pero se cambia el 0 por *.

B - Indica un espacio en blanco.

- ó + - Indican la aparición del signo correspondiente.

Para indicar la longitud del campo se puede repetir el símbolo tantas veces como longitud tenga o expresarla entre paréntesis, es decir para definir una variable alfanumérica de 10 caracteres se pondría: PIC X(10) o PIC XXXXXXXXXX. Los valores S y V solo pueden aparecer una vez por cada variable.

Ejemplos del WORKING STORAGE SECTION serán los siguientes:

WORKING-STORAGE SECTION.

01 DOMICILIO.

02 TIPO PIC XX.

02 NOMBRE PIC X(20).

02 NUMERO PIC 9(4).

WORKING-STORAGE SECTION.

77 FECHA PIC 9(8).

77 FECHA-EDIT PIC ZZ/ZZ/ZZZZ.

77 IMPORTE PIC S9(8)V99.

77 IMPORTE-EDIT PIC ZZ.ZZZ.ZZZ,ZZ-.

01 ESTADO PIC 9.

88 SOLTERO VALUE 1.

88 CASADO VALUE 2.

88 DIVORCIADO VALUE 3.

Suponiendo que el valor de FECHA es 01111998 (1 Nov de 1998) y FECHA-EDIT es el mismo, éste último se representaría: 01/11/1998.

Si IMPORTE es 12815V37 en negativo e IMPORTE-EDIT el mismo, éste se representaría: 12.815,37-.

VALUE esta palabra a continuación del PIC indica el valor inicial que contendrá la variable hasta que éste sea modificado.

WORKING-STORAGE SECTION.

01 RAYA PIC X(10) VALUE "-----".

01 MIWEB PIC X(30) VALUE "www.jlbv.com".

En este caso tenemos una variable llamada RAYA que tiene 10 guiones seguidos. Y Otra variable llamada MIWEB que indica www.jlbv.com

REDEFINES esta cláusula se utiliza para dar mas de un nombre y formato a un mismo campo. Este debe de ir a continuación del nombre de campo y antes del nombre del campo a que hace referencia, deben de estar en el mismo nivel y uno a continuación del otro en el orden de declaraciones.

WORKING-STORAGE SECTION.

01 DIAS PIC X(21) VALUE "LUNMARMIEJUEVIESABDOM".

02 DIA REDEFINES DIAS.

02 DIA PIC XXX OCCURS 7 TIMES.

La cláusula OCCURS nos permite definir una variable día que se va a repetir 7 veces tomando los valores de DIAS debido a la cláusula REDEFINES.

JUST la justificación de los valores de los campos suele ser a la izquierda para los alfanuméricos y a la derecha para los numéricos, si en cambio queremos cambiar este orden tendremos que incluir ésta cláusula.

WORKING-STORAGE SECTION.

01 NOMBRE PIC X(10).

01 NOMBRE1 PIC X(10) JUST RIGHT.

USAGE con ésta cláusula determinamos el formato en que se guarda el contenido de las variables (numéricas, ya que las alfanuméricas siempre ocuparán un byte por cada carácter). Con todos los campos se puede operar (obviamente) pero solo los que se definan como DISPLAY se podrán editar directamente. Tiene varias posibilidades:

DISPLAY, es la forma por defecto e indica que cada dígito ocupará un byte, es la que se toma por defecto y la que memoria ocupa.

BINARY, COMP-1, COMP-3, COMP-6, son diferentes formas de compactación de los datos. COMP-6 (la mas usual) guarda dos dígitos en cada byte, COMP-3 es igual pero admite signo el cual iría en los cuatro últimos bites del último byte. Las restantes formas se utilizan menos.

WORKING-STORAGE SECTION.

01 IMPORTE PIC 9(8). Ocupa 8 bytes uno por cada dígito.

01 FECHA PIC 9(8) COMP-6. Ocupa 4 bytes, uno cada 2 dígitos.

01 PRECIO PIC S9(8)V99 COMP-3. Ocupa 6 bytes, uno para los dos decimales, cuatro para la parte entera y uno para el signo.

Cuando el tamaño es impar el signo no ocupa ya que comparte byte con el último dígito, el punto decimal tampoco ocupa espacio, ya que solo indica su posición. Estas tres formas que he utilizado son las mas comunes, yo por ejemplo como las capacidades de los ordenadores son tan grandes ahora, cuando defino las variables en la WORKING nunca utilizo ninguna compactación es decir no utilizo la cláusula USAGE (que por cierto y como habréis visto se puede omitir) y por defecto toma DISPLAY (un byte por carácter).

LINKAGE SECTION. En ésta sección se declaran las variables de igual forma solo que las que aquí declaremos nos van a servir de enlace para pasar información a otro programa que será llamado por el principal.

La DATA DIVISION nos sirve para tener todas nuestras variables bien definidas, ya sean independientes o que formen parte de algún fichero para poder operar con ellas en la PROCEDURE DIVISION. Me acuerdo de cuando empezamos a estudiar que para otros lenguajes no era necesaria la declaración de variables previamente y en cambio ahora en la mayoría de los lenguajes se exige que se declaren, eso significa que el Cobol no iba mal encaminado.

Tenemos que tener siempre muy claro que con cualquier campo podemos hacer lo que queramos, por ejemplo si en un registro de 120 caracteres nosotros en un programa solo vamos a utilizar los 40 primeros podemos definir todo lo restante como un campo FILLER y listo o viceversa si un campo de un registro lo tenemos definido como alfanumérico de 30 en un programa necesitamos los 10 primeros caracteres por un lado y los veinte restantes por otro, se subdivide para ese programa y no pasa nada.

Espero que haya quedado por lo menos medio clara la explicación de ésta tercera DIVISION, evidentemente con la práctica es con lo que mas vamos a aprender siempre claro está que tengamos al menos unas nociones mínimas.

Tenemos que tener en cuenta que como en todos los lenguajes, ya sean de programación o de habla (español, inglés, francés) son muchas las opciones que nos ofrecen pero al final siempre utilizamos las que más nos gustan o las que consideramos más útiles.

SCREEN SECTION.

En ésta sección podremos describir los atributos y campos a utilizar en las pantallas.

PROCEDURE DIVISION.

Todas las divisiones hasta esta se encargaban de definir las características del programa. Sin embargo, esta división considerada como el verdadero corazón del programa, es donde se programan todos los procedimientos.

Aquí se definen todas las funciones del programa, y el ordenador ejecutará todas de forma secuencial de tal forma que el algoritmo tenga una estructura más o menos lógica con la cual se obtendrán unos resultados correctos o incorrectos dependiendo de si el programa está hecho bien o no.

Un resumen de una PROCEDURE DIVISION sería la siguiente:

PROCEDURE DIVISION (USING Variable, Variable ...).

DECLARATIVES.

Nombre-sección SECTION.

USE AFTER ERROR PROCEDURE ON tipo.

Nombre-parafo.

Sentencias.

.....

END DECLARATIVES.

Nombre-sección SECTION.

Nombre-parafo.

Sentencias.

.....

PROCEDURE DIVISION (USING Variable, Variable ...).

Cuando especificamos USING en la línea de PROCEDURE DIVISION, después deberemos de dar los nombres de variables que hayamos definido en la LINKAGE SECTION, para compartir en el programa, lo que nos indicará que éste ha sido llamado por otro programa y que esas variables traerán un valor procedente del programa llamador, que a su vez utilizó la instrucción CALL con las mismas variables.

DECLARATIVES, es una sección dentro de la PROCEDURE que nos va a servir para controlar los posibles errores en cuanto al manejo de ficheros se refiere. La línea de DECLARATIVES, (si se va a utilizar) deberá de ir siempre a continuación de la línea de PROCEDURE DIVISION. Después de subdividir en tantas secciones como opciones de error tengamos, éstas pueden ser definidas por archivo o bien forma de

apertura, es decir podremos controlar los errores que nos lleguen de un fichero en concreto o de todos aquellos que hayan sido abiertos de de igual forma, esto se especifica en la línea USER AFTER ERROR PROCEDURE ON tipo, pudiendo ser tipo, el nombre del archivo o su modo de apertura (INPUT, OUTPUT, I-O, EXTEND. A continuación irían los párrafos con sus respectivas instrucciones a realizar en caso de error. Se pondrían tantos párrafos y secciones como quisiéramos controlar, siempre teniendo en cuenta que ésta se acaba cuando se indique END DECLARATIVES.

Si no quisiéramos utilizar ésta sección, podríamos de igual manera controlar los errores en nuestro programa preguntando siempre por la variable de error de cada fichero que se definió como FILE STATUS en la FILE-CONTROL.

Nombre-sección SECTION, a partir de aquí incluiremos todas las instrucciones necesarias para la correcta ejecución del programa.

Ya sabemos que Cobol es un lenguaje estructurado, pues bien no pensemos que la Procedure va a ser un caos de instrucciones escritas secuencialmente, no, en ella podremos definir tantas Secciones (SECTION) y Párrafos como queramos para organizar mejor las instrucciones y para delimitar acciones concretas, eso si siempre se ejecutarán secuencialmente, excepto cuando encuentre algún verbo de bifurcación como GO, PERFORM que haciendo referencia a esos nombres de párrafo harán que se rompa la secuencia lógica de ejecución.

Si se especifica (NOT) se da a entender lo contrario de la comparación, con lo que no mayor que puede ser igual que menor o igual que.

También podemos utilizar para las condiciones complejas los operadores lógicos, AND, OR y NOT según se requieran, todas las condiciones que incluyan AND han de ser correctas para que pase la condición, si se utiliza OR solo alguna de ellas ha de ser correcta, cuando se utilice NOT no podrá ser correcta para que la condición sea válida.

Sentencias básicas.

Hay dos sentencias básicas, ACCEPT y COPY, ambas realizan funciones propias sin llamar a otras.

ACCEPT

Nos permite introducir valores en memoria provenientes del teclado.

`ACCEPT variable CLAUSULAS ... ON EXCEPTION variable instrucción.`

Aunque la sentencia podría ser la siguiente para obtener el día o la hora del ordenador:

`ACCEPT variable FROM (DATE, DAY, DAY-OF-WEEK, TIME, CENTURY-DATE, CENTURY-DAY, ESCAPE-KEY, ENVIRONMENT variable de entorno)`

Según la opción escogida, así será el valor que contendrá la variable usada, una vez completada la sentencia. Veamos cada una de las opciones:

DATE, devuelve la fecha en formato AAMMDD, por lo que la variable debe de estar definida con PIC 9(6).

CENTURY-DATE, igual que DATE, pero acepta la fecha con 8 dígitos en formato SSAAMMDD. Definir con PIC 9(8).

DAY, devuelve el año y el día del año en que estamos con el formato AADDD, siendo el valor 1, para el 1 de Enero y el 365 el 31 de Diciembre en un año no bisiesto. Debe de estar definida con PIC 9(5).

CENTURY-DAY, igual que DAY, pero acepta 4 dígitos para el año, quedando el formato AAAADDD. Definir con PIC 9(7).

DAY-OF-WEEK, devuelve un dígito que indica el día de la semana, siendo 1 el Lunes, 2 el Martes, y 7 el Domingo. Aquí, la variable debe de estar definida como PIC 9.

TIME, devuelve la hora con formato HHMMSSCC, la variable debe de estar como PIC 9(8).

ESCAPE-KEY, devuelve el código de la tecla de excepción pulsada, debe de estar definida con PIC 99 y según el compilador los valores pueden ser distintos, pero los básicos suelen ser los mismos, por ejemplo: ENTER=13, FLECHA ARRIBA=52, FLECHA-ABAJO=53 y las teclas de función desde F01 hasta F8 tomarían los valores del 1 al 8, TAB=9.

ENVIRONMENT "Variable de entorno" nos devuelve el valor que dicha variable tenga asignado, normalmente éstas variables se asignan en el fichero de configuración del compilador o desde una variable del sistema. Puede ser muy útil para dar capacidades a usuarios diferentes por ejemplo. Tener en cuenta que la variable de entorno debe de ir entre comillas para que sea reconocida.

DISPLAY

Es la instrucción que usaremos como salida de datos en pantalla. Con ella mostraremos cualquier texto, cualquier variable, cualquier constante o cualquier valor.

DISPLAY variable, literal **CLAUSULAS** ...

Casi todas las cláusulas de ACCEPT son idénticas a las usadas en DISPLAY, por eso vamos a indicar si aparecen las siguientes en ACCEPT, en DISPLAY o en ambas:

LINE variable, constante numérica, (ACCEPT Y DISPLAY) le indicamos la línea donde se aceptará el campo, puede ser una variable o un número, tendremos siempre en cuenta que la línea 1 será la primera de arriba de la pantalla, normalmente podremos trabajar con 24 líneas. Si no se especifica tomará la referencia de la siguiente línea en la que estemos.

COLUMN, **COL**, **POSITION** variable, constante numérica, (ACCEPT Y DISPLAY) aquí le indicamos la columna dentro de la línea y sus valores pueden ser desde 1 (izquierda) hasta 80 (derecha) de la pantalla. En algunos terminales es posible capturar 132 columnas. Si no se especifica tomará el valor 1.

BEEP, **NO BEEP**, (ACCEPT Y DISPLAY) con ésta opción le indicamos si queremos o no que emita un pitido al llegar al comando. Por defecto sonará.

BLINK, (ACCEPT Y DISPLAY) si aparece ésta cláusula, el campo que aceptemos parpadeará.

HIGH, **LOW**, **OFF**, (ACCEPT Y DISPLAY) le indicamos el nivel de luminosidad con que se acepta el campo, solo puede ir especificado uno: **HIGH** con brillo, **LOW** sin brillo, **OFF** hace que los datos que introducimos no se vean, se utiliza cuando queremos aceptar una contraseña o alguna clave, en la pantalla no aparecen los datos que vamos introduciendo. Si no indicamos ninguna el sistema coge **HIGH** por defecto.

TAB, (ACCEPT) la presencia de ésta cláusula hace que no salte al siguiente campo hasta que no pulsemos **ENTER**. Por defecto al introducir el número de caracteres o dígitos del campo salta automáticamente.

TIME variable, constante numérica, (ACCEPT) conseguiremos que la espera para introducir el campo sea limitada y vendrá dada por el valor que le asignemos. El valor se indica en centésimas de segundo.

CURSOR variable, (ACCEPT) si especificamos ésta cláusula, el sistema guardará en la variable la posición donde se ha terminado la introducción de los datos. Por ejemplo si aceptamos un campo de **PIC X(40)** y solo rellenamos 9 caracteres, el valor de la variable asociada a **CURSOR** será de 10.

ERASE, haremos que se borra toda la pantalla.

ERASE EOL, (ACCEPT Y DISPLAY) le indicamos que borre la línea sobre la que estamos desde la posición del **ACCEPT**.

ERASE EOS, (ACCEPT Y DISPLAY) le indicamos que borre toda la página desde la posición de la línea en la que estamos.

PROMPT carácter, (ACCEPT) conseguimos que rellene con algún carácter específico o por defecto el cursor bajo () la longitud del campo que vamos a aceptar. Es bastante útil ya que el usuario puede ver claramente la longitud del campo.

ECHO, (ACCEPT) con ésta cláusula hacemos que vuelva a "pintar" el contenido del campo después de aceptado. Es decir si le hemos puesto PROMPT ECHO, mientras se acepta saldrán los guiones bajos para indicar el tamaño pero al salir del campo se volverá a mostrar y estos guiones desaparecerán.

REVERSE, (ACCEPT Y DISPLAY) conseguimos que el campo que aceptamos se vea en video inverso, si normalmente es blanco sobre negro, se hará negro sobre blanco, y si tenemos otros colores, pues en su color inverso.

SIZE variable, constante numérica, (ACCEPT) variamos el tamaño de la variable que teníamos declarado. Por ejemplo si definimos una variable con PIC X(40) y le indicamos al aceptarla SIZE 10, solo aceptará 10 caracteres para esa variable.

CONVERT, (ACCEPT Y DISPLAY) con ésta cláusula conseguimos que si estamos aceptando un campo numérico o alfabético, e introducimos un valor no aplicable, lo ignore. Por ejemplo si introducimos una letra en un campo numérico será ignorada si CONVERT aparece.

UPDATE, (ACCEPT) hace que el valor actual del campo aparezca y al aceptar podamos modificar sobre su contenido actual. Es muy aconsejable cuando estamos modificando datos.

CONTROL tipo de control, (ACCEPT Y DISPLAY) ésta cláusula es bastante amplia y en ella podemos atribuir otras opciones como caracteres gráficos, colores, etc.. Su uso puede variar según los compiladores. Mas abajo encontrareis mas información.

ON EXCEPTION variable instrucción (ACCEPT) si al aceptar el campo introducimos una de las teclas que causa excepción, es decir que no es reconocida dentro de la tabla de caracteres admitida, el valor de la tecla se guardará en la variable que definamos y a continuación podremos realizar una instrucción. Por ejemplo si estamos aceptando un campo con PIC X(40) y cuando estamos metiendo su valor pulsamos la tecla escape, si tenemos ésta cláusula puesta la variable tomará el valor 27 (correspondiente a la tecla escape) y a continuación se ejecutaría la instrucción que hubiéramos puesto. Decir que la variable debe de estar definida con PIC 9 COMP-1.

UPPER, LOWER, (ACCEPT) con éstas opciones obligamos a que el valor del campo aceptado esté en Mayúsculas (UPPER) o Minúsculas (LOWER).

Color	Letra	Fondo
Negro	1	32
Azul	2	64
Verde	3	96
Celeste	4	128
Rojo	5	160
Morado	6	192
Marrón	7	224
Blanco	8	256

COLOR, (ACCEPT y DISPLAY) cláusula propia de AcuCobol con la que conseguimos aceptar el campo con un determinado color, dependiendo de una tabla para asignación de colores. El color se consigue sumando los colores de fondo y de letra. Por ejemplo, si queremos poner fondo azul y escribimos 72, deducimos ya que 72 menos 64 son 8 que el color de la letra es el blanco, $TOTAL = LETRA + FONDO$.

REQUIRED, (ACCEPT) con ésta opción obligamos al usuario a que no deje el campo en blanco e introduzca cualquier dato.

FULL, (ACCEPT) obligamos a que una vez empecemos a introducir un campo, este no se acepte hasta que esté completo, es decir si aceptamos una variable con PIC X(30), mientras no introduzcamos los 30 caracteres no nos dejará seguir.

COPY

Sirve para ahorrar espacio en el disco, usando el código escrito en el fichero de forma temporal cuando se utiliza esta instrucción.

COPY fichero1.

MOVE

Es la instrucción que usaremos para enviar datos de una variable a otra u otras. Lo que en realidad hace es que la una variable adquiera un valor determinado, ya sea procedente de otra variable o bien desde un valor fijo o constante.

MOVE *variable*, *valor* **TO** *variable*, *variable*, *variable*,

Con este formato, todas las variables anteriores al TO tomarán el valor que tenemos en la variable o el valor indicado después de MOVE.

Los campos numéricos siempre se van a alinear a la derecha, respetando la posición del punto decimal si lo hubiera. Si la variable que recibe el campo es mas pequeña, evidentemente se perderán los que no quepan y si es mas grande el resto se pondrá a ceros.

- Si además el campo al que se mueven los datos es de edición, al hacer el paso del valor, este a su vez se formateará con la edición declarada en la WORKING.
- Si son alfanuméricos la alineación se efectuará a la izquierda a menos que se haya especificado en la WORKING, al definirla, una justificación a la derecha (JUST RIGHT). Al igual que en los numéricos si es mas pequeño se perderán los caracteres que no quepan y si es mas grande el resto irá relleno de espacios en blanco.

MOVE CORR *Identificador1* **TO** *Identificador2*

Agregando CORR a la instrucción conseguimos mover de una sola vez un valor entre identificadores siempre que los campos que contengan tengan el mismo nombre. Estos identificadores no pueden ir en niveles 66, 77 ni 88. El efecto es el mismo que si hiciéramos tantos MOVE normales como campos iguales tuviera el identificador. No es muy usual, pero si hay casos en los que puede ser razonable su uso.

INITIALIZE

Se utiliza para inicializar variables según su descripción, es decir pondrá a ceros todas las variables numéricas o de edición y a espacios en blanco las alfabéticas y alfanuméricas. No funciona con campos definidos como FILLER, (evidente). Y puede ser muy útil para inicializar tablas completamente cuando nos referimos al nivel más alto de la misma.

INSPECT

Esta sentencia se utiliza para contar, reemplazar o contar y reemplazar caracteres o grupos de caracteres dentro de un campo. Se puede contar las veces que aparece un carácter, o cambiar todos esos caracteres por otros, entre otras cosas.

INSPECT *campo1*

TALLYING *variable1* **FOR (CHARACTERS)**

((BEFORE/ AFTER) INITIAL) *Cadena1*

(ALL)(LEADING) *Cadena2 ...*

Este formato es el utilizado para contar el número de veces que aparece *Identificador1* en el *campo1* y guardará el valor en la *variable1* que previamente hayamos definido en la **WORKING**.

WORKING-STORAGE SECTION.

77 *TEXTO* **PIC** *X(15)* **VALUE** "PAGINA DE COBOL".

77 *CONTA* **PIC** *9(8)*.

...

PROCEDURE DIVISION.

INICIO.

INSPECT *TEXTO* **TALLYING** *CONTA* **FOR** *CHARACTERS*.

...

El valor de *CONTA* será de 15 que son los caracteres que tiene la variable *TEXTO*.

INSPECT *TEXTO* **TALLYING** *CONTA* **FOR** *ALL "A"*.

...

El valor de *CONTA* será de 2 que son las veces que aparece la letra A en la variable *TEXTO*.

INSPECT *TEXTO* **TALLYING** *CONTA* **FOR** *LEADING "A"*.

...

El valor de CONTA será de 0 porque no aparece ninguna A en el primer carácter de la variable TEXTO.

```
INSPECT TEXTO TALLYING CONTA FOR ALL "A"
BEFORE INITIAL "N".
```

...

El valor de CONTA será de 1 que son las veces que aparece la letra A en la variable TEXTO hasta la aparición del carácter N.

```
INSPECT TEXTO TALLYING CONTA FOR ALL "A"
AFTER INITIAL "G".
```

...

El valor de CONTA será de 1 que son las veces que aparece la letra A en la variable TEXTO, empezando a contar desde el carácter G.

Otro formato para la instrucción que nos permitirá reemplazar será:

```
INSPECT campo1
REPLACING variable1 CHARACTERS BY Cambio1
(( BEFORE/ AFTER) INITIAL) Cadena1
(ALL)(LEADING)( FIRST) Cadena2...
```

Con este formato podemos cambiar caracteres de Campo1. Su funcionamiento es igual que el anterior formato solo que en vez de contar reemplaza. Se ha incluido solo FIRST, que indicaría que solo se reemplazaría la primera vez que coincidieran las condiciones. El tamaño de la sustitución debe de ser igual al tamaño sustituido, ya que la variable campo1 no puede cambiar su tamaño.

```
INSPECT campo1
CONVERTING Identificador1 TO Identificador2
(( BEFORE/ AFTER) INITIAL) Cadena1 ...
```

Con este formato convertimos los caracteres que se especifiquen en identificador1 por los que pongamos en identificador2, respetando el orden.

...

WORKING-STORAGE SECTION.

77 TEXT PIC X(15) VALUE "PAGINA DE COBOL".

...

PROCEDURE DIVISION.

INICIO.

INSPECT TEXT CONVERTING "AO" TO "12".

...

El valor de TEXT será "P1GIN1 DE C2B2L", convertirá todas las A por 1 y todas las O por 2.

```
INSPECT  TEXT      CONVERTING  "ABC DEFG HIJKLMNÑO PQ RSTUVWXYZ"  TO
                                         "a b c d e fg hijklm nño p q rstuv wxyz".
```

...

El valor de TEXT será "pagina de cobol" ya que ha convertido todas las letras mayúsculas por minúsculas.

STRING

Se utiliza para unir o concatenar campos o partes de estos y el resultado almacenarlo en otro campo. En la unión se pueden incluir tanto variables como literales o constantes de texto.

```
STRING c a m p o 1, l i t e r a l 1
      DELIMITED BY (c a m p o 2, l i t e r a l 2)(SIZE)
      INTO C a m p o 3
      (WITH POINTER I d e n t i f i c a d o r 1)
      (ON OVERFLOW S e n t e n c i a 1)
      (NO T O N O V E R F L O W S e n t e n c i a 2)
```

DELIMITED BY, indica hasta donde vamos a "coger" del campo para concatenar sin contar ese carácter o cadena que se especifique en campo2 o literal2, es decir si tenemos un campo con un valor = "HOLA" y especificamos DELIMITED BY "L" a la hora de la concatenación nos hubiera cogido solo el HO, ya que al encontrarse la primera L hubiera parado.

SIZE, indica que se pasará todo el contenido del campo1 o literal1 sin limitaciones.

INTO, con esto indicamos en que variable se guardará el resultado, campo3.

WITH POINTER, si incluimos esta cláusula el valor de identificador1 será en la posición en que empezará a contener datos la variable que recibe el STRING. Ese identificador1 debe estar definido como binario. Por defecto el valor es 1.

ON OVERFLOW, se ejecutaría Sentencia1 si hubiera habido un error al hacer la concatenación, por ejemplo si se especifica Identificador1 con un valor superior al tamaño del Campo3.

NOT ON OVERFLOW, se ejecutará Sentencia2 si no existe error en la operación.

WORKING-STORAGE SECTION.

01 LAFECHA.

02 FILLER PIC X(7) VALUE "HOY ES ".

02 LDIA PIC Z9.

02 FILLER PIC X(4) VALUE " DE ".

02 LMES PIC X(10).

02 FILLER PIC X(4) VALUE "DE ".

02 LANIO PIC 9999.

01 FECHA.

02 DIA PIC 99 VALUE 22.

02 MES PIC 99 VALUE 06.

02 ANIO PIC 9999 VALUE 2001.

01 CONSTRING PIC X(40).

01 TABLAMES.

02 FILLER PIC X(30) VALUE "ENERO FEBRERO MARZO ".

02 FILLER PIC X(30) VALUE "ABRIL MAYO JUNIO ".

02 FILLER PIC X(30) VALUE "JULIO AGOSTO SEPTIEMBRE".

02 FILLER PIC X(30) VALUE "OCTUBRE NOVIEMBREDICIEMBRE ".

01 LATABLA REDEFINES TABLAMES.

02 TMES PIC X(10) OCCURS 12 TIMES.

01 PUNTO PIC 9(4) BINARY.

...

PROCEDURE DIVISION.

INICIO.

MOVE DIA TO LDIA.

MOVE TMES (MES) TO LMES.

MOVE ANIO TO LANIO.

MOVE 4 TO PUNTO.

STRING "HOY ES " DIA " DE " DELIMITED BY SIZE

- TIMES (MES) DELIMITED BY " "
- " DE " ANIO DELIMITED BY SIZE INTO CONSTRING
- WITH POINTER PUNTO.
- ...

UNSTRING

Hace exactamente lo contrario que hacía STRING, es decir divide el contenido de un campo en otros.

UNSTRING campo 1, literal

DELIMITED BY (campo 2, literal)(ALL)

OR (campo 2, literal)(ALL)

(Se puede repetir de nuevo)

INTO Campo 3, Campo 4,

(DELIMITER Identificador 1)

(COUNT Identificador 2)

(Se puede repetir de nuevo)

(WITH POINTER Identificador 3)

(TALLYING Identificador 4)

(ON OVERFLOW Sentencia 1)

(NOT ON OVERFLOW Sentencia 2)

DELIMITED BY, indica el límite hasta donde vamos cogiendo el campo 1 para partirlo. Igual que en STRING, solo que con la función a la inversa.

OR, es igual que DELIMITED y se utiliza si hay varios delimitadores sobre los que buscar.

INTO, indica en que campo o campos se guardará la información que vaya fragmentando.

DELIMITER, va a contener en cada caso el elemento separador, si hemos incluido en DELIMITED varios, Identificador 1 guardará el carácter que de los elegidos ha sido el causante de la fragmentación.

COUNT, cuenta el número de caracteres incluidos en la fragmentación.

DELIMITER y COUNT, se podrán usar si se ha especificado DELIMITED. Podemos usar tantos DELIMITER y COUNT como campos se vayan a crear en la fragmentación.

TALLYING, si especificamos esta opción la instrucción nos guardará en Identificador4 el número de campos que se han utilizado en la fragmentación.

POINTER, indica desde que posición va a ser examinado el campo que desea desfragmentar, por defecto su valor es 1, es decir desde el primer carácter.

ON OVERFLOW, se ejecutaría Sentencia1 si hubiera habido un error al hacer la operación.

NOT ON OVERFLOW, se ejecutará Sentencia2 si no existe error en la operación.

Sentencias de control.

IF ... THEN ... ELSE

Esta es la sentencia de control de más importancia ya que permite dependiendo de una condición elegir un camino u otro. Su sintaxis es la siguiente:

```
IF Condición THEN {Sentencia1 / NEXT SENTENCE} ELSE {Sentencia2 / NEXT SENTENCE}
```

Las condiciones se consiguen enfrentando dos variables mediante las condiciones <, > o =. Como condiciones especiales nos podemos encontrar con:

```
IF Identificador IS [NOT] {POSITIVE / NEGATIVE / ZERO}.
```

Para saber si un identificador es o no positivo, negativo o neutro.

```
IF Identificador IS [NOT] {ALPHABETIC / NUMERIC}.
```

Para saber si el identificador es o no letra o número.

PERFORM

Esta sentencia nos permite ejecutar un párrafo en cualquier momento, y con la opción TIMES lo podemos realizar varias veces consecutivas.

```
PERFORM Párrafo1 [THRU Párrafo2] {Identificador / Entero} TIMES
```

Si quisiéramos reiterar la ejecución del párrafo especificado hasta que se cumpla la condición impuesta.

```
PERFORM Párrafo1 [THRU Párrafo2] VARYING Identificador1 FROM Identificador2 BY  
Identificador3 UNTIL Condición.
```

STOP

Indica el final del programa y su aparición hace volver al sistema operativo.

```
STOP RUN.
```

Si quisiéramos que nos indicara un valor al terminarse el programa usaríamos:

```
STOP Literal
```

Instrucciones Matemáticas.

Debido a que el lenguaje Cobol fue concebido para la gestión de grandes cantidades de datos y a resolver problemas de tipo comercial y de administración, no se incluyeron dentro de sus especificaciones verbos que nos pudieran ayudar a resolver cálculos complejos como integrales, trigonometría, raíces cuadradas, etcétera, sino simplemente las orientadas a los cálculos básicos, suma, resta, multiplicación y división que son las que vamos a ver a continuación:

ADD

Añade el valor de dos identificadores o más a otro identificador. Este valor puede estar o no redondeado:

ADD Identificador1, Identificador2 [, Identificador3 ...]TO IdentificadorX [ROUNDED] [IdentificadorY [ROUNDED] ...] [ON SIZE ERROR Declaración].

Otra opción permite guardar la suma de dos o más identificadores a otro:

ADD Identificador1, Identificador2 [, Identificador3 ...] GIVING Identificador X [ROUNDED].

Si queremos sumar variables de grupo, usaremos otra sentencia de tipo ADD:

ADD {CORRESPONDING / CORR} Identificador1 TO Identificador2 [ROUNDED] [ON SIZE ERROR Declaración].

Un ejemplo de instrucción de ADD sería el siguiente:

WORKING-STORAGE SECTION.

01 VALORES.

02 UNO PIC 99 VALUE 10.

02 DOS PIC 9(6) VALUE 280.

02 TRES PIC 9(4) VALUE 540.

01 OTROS

OTRO1 PIC 9.

02 DOS PIC 9(6) VALUE 110.

01 RESULTADO PIC S9(8) VALUE 10.

01 RESTO PIC 99.

PROCEDURE DIVISION.

INICIO.

Tres afirmaciones se podrían decir del resultado:

ADD UNO TRES 5 TO RESULTADO .

Esta instrucción suma UNO, TRES y el valor 5 al valor contenido en RESULTADO, donde se guarda.

ADD UNO TRES 5 GIVING RESULTADO .

Esta instrucción suma UNO, TRES y el valor 5, guardando el resultado en la variable RESULTADO.

ADD CORR VALORES TO OTROS .

Esta instrucción incrementa las variables que aparecen en ambos por medio de su suma, en este ejemplo se suma DOS de VALORES de DOS de OTROS, donde se guardan los resultados.

SUBTRACT

Funciona exactamente igual que la instrucción ADD en cada una de sus tres modalidades con la diferencia de que en este caso lo que hace es que va restando.

SUBTRACT UNO TRES 5 FROM RESULTADO .

Se encargará de restar las tres variables al valor de RESULTADO, guardándose en esta variable.

SUBTRACT UNO 5 FROM TRES GIVING RESULTADO .

Restamos en este caso a TRES las variables UNO y el valor 5, guardándolo en la variable RESULTADO.

SUBTRACT CORR VALORES FROM OTROS .

Restamos las variables que tengan el mismo nombre en VALORES y OTROS.

MULTIPLY

Esta instrucción nos permite multiplicar dos valores y guardar el resultado en el segundo de ellos:

MULTIPLY Identificador1 BY Identificador2 [ROUNDED] [ON SIZE ERROR Declaración]

Una segunda posibilidad sería la de multiplicar dos valores y guardarlas en un tercer identificador.

MULTIPLY Identificador1, Identificador2 GIVING Identificador3 [ROUNDED] [ON SIZE ERROR Declaración]

También se pueden multiplicar todos los valores que están acordes con un nombre:

MULTIPLY {CORRESPONDING / CORR} Identificador1 BY Identificador2 [ROUNDED] [ON SIZE ERROR Declaración]

MULTIPLY 5 FROM RESULTADO .

Multiplica 5 por el valor de RESULTADO, guardando el resultado en RESULTADO.

MULTIPLY 5 BY TRES GIVING RESULTADO .

Multiplica 5 por el valor de TRES guardándolo en RESULTADO.

DIVIDE

Funciona exactamente igual que la instrucción MULTIPLY solo que dividiendo. Tendríamos que sustituir la palabra MULTIPLY por DIVIDE para obtener estos resultados.

DIVIDE 10 INTO TRES.

Divide 10 entre TRES y guarda el resultado en TRES.

DIVIDE 7 INTO TRES GIVING RESULTADO REMAINDER RESTO.

Divide 7 entre TRES, guardando el resultado en RESULTADO y el resto en RESTO.

COMPUTE

Esta orden nos permite realizar todos los cálculos aritméticos posibles en una sola instrucción, utilizando los operadores +(suma) -(resta) *(multiplicación) /(división) **(potenciación), además de utilizar paréntesis para especificar mejor la operación a realizar.

COMPUTE Variable (ROUNDED) = ExpresiónAritmética (ON SIZE ERROR) Instrucción

C O M P U T E R E S U L T A D O = (D O S O F V A L O R E S ** 2) / 1 0 0 .

Manipulación de tablas.

Si usamos tablas somos capaces de reducir una gran cantidad de información agrupada por su rasgo en común. Las tablas

El ejemplo más usual de tabla suele ser la de los meses. El ordenador siempre nos devuelve un número del 1 al 12, sin embargo intentamos mostrar este número por pantalla siempre que podemos con los nombres que van desde Enero hasta Diciembre.

```
01 TABLA-MES
    02 ELEMENTO-TM OCCURS 12.
        03 COD-MES PIC 99.
        03 LIT-MES PIC X(10).
```

La forma básica de creación de tablas es la siguiente:

WORKING STORAGE SECTION.

```
01 VALORES-TABLA-MES.
    02 FILLER PIC 99 VALUE 1.
    02 FILLER PIC X (10) VALUE "Enero".
    02 FILLER PIC 99 VALUE 2.
    02 FILLER PIC X (10) VALUE "Febrero".
```

Una forma más reducida de definir la tabla sería:

WORKING STORAGE SECTION.

```
01 VALORES-TABLA-MES.
    02 FILLER PIC X(13) VALUE "01 Enero".
    02 FILLER PIC X(13) VALUE "02 Febrero".
```

O la forma más reducida de todas:

WORKING STORAGE SECTION.

```
01 VALORES-TABLA-MES.
    02 FILLER PIC X (156) VALUE "01 ENERO 02 FEBRERO ... ".
```

Pero la forma de definirlo sería la siguiente:

WORKING STORAGE SECTION.

01 TABLA-MES REDEFINES VALORES-TABLA-MES.

02 ELEMENTO-TM OCCURS 12.

03 COD-MES PIC 99.

03 LIT-MES PIC X (10).

Un ejemplo completo de ejercicio hecho con tablas es el siguiente:

01 TABLA.

02 TRABAJADORES OCCURS 100 TIMES

- ASCENDING KEY NO MBRE

- INDEXED BY SUB1.

03 NOMBRE PIC X(30).

03 NIF PIC X(10).

...

PROCEDURE DIVISION.

...

BUSCAR.

SEARCH ALL ELEMENTS ATEND GO NO EXIS

- WHEN NOMBRE (SUB1) = "PACO PEREZ" GO EXISTE

NO EXIS.

DISPLAY "TRABAJADOR INEXISTENTE" LINE 1.

GO FIN.

EXIS.

DISPLAY "EL NIF DEL TRABAJADOR ES ." LINE 2.

DISPLAY NIF (SUB1) LINE 2 COL 30.

...

Ficheros de entrada y salida.

Cobol no solo es un lenguaje de programación, sino que además de las instrucciones de programación tiene un gran administrador de ficheros. Esta es una gran diferencia con el resto de lenguajes de programación, ya que el resto de lenguajes depende de bases de datos y herramientas externas.

Cobol tiene una diferencia con el resto de bases de datos, y es que al estar hecho el lenguaje y la gestión de datos en el mismo programa, la fiabilidad de estos es mayor al del resto de programas.

A tomar muy en cuenta es la facilidad de creación de archivos indexados. A diferencia de los otros lenguajes de programación. Los ficheros que se hacen en este lenguaje son bastante más eficaces que los que se hacen en el resto de lenguajes.

Los ficheros indexados de Cobol, pueden dividir el fichero en dos archivos físicos, uno para las claves y otro para los datos, otros en cambio lo guardan todo en uno mismo, pero eso no significa que no lo haga igual, sino que al usuario solo le muestra un fichero físico, que puede resultar mas cómodo.

Cuando se graba un nuevo registro, éste se ordena automáticamente en orden ascendente por la clave principal. Luego podremos modificar tantas veces como deseemos los datos, pero la clave nunca se podrá alterar. Si queremos cambiar la clave, tendremos que borrar el registro y grabar otro con la clave deseada. De esa manera Cobol se asegura el perfecto funcionamiento de su sistema de índices.

La parte de índices es como una tabla con las posiciones de memoria de los datos que le corresponden. Si el fichero está abierto en modo I-O y se produce una salida brusca del programa o un corte de luz, puede ocurrir que esa información sobre los datos que corresponden a cada índice se alteren y de ahí el famoso y terrible error 98. Por eso se aconseja siempre leer el fichero abierto sólo como Input (modo de lectura) y abrirlo como I-O (modo de escritura) sólo en el preciso momento de grabar o borrar su contenido.

Lo realmente importante para Cobol cuando crea un fichero, es el tamaño del registro y el de la clave en bytes. El resto le da igual, incluso la estructura se puede definir de maneras totalmente diferentes.

Para Cobol, si generamos un fichero con un registro de 128 posiciones, eso es lo que guarda, si nosotros le indicamos que el nombre ocupa 40 y comienza en la posición 18 perfecto, pero si la próxima vez le indicamos otra estructura el también la aceptará.

Una posibilidad que ofrece COBOL es la posibilidad de incluir claves alternativas, las cuales nos dan la posibilidad de acceder al fichero indexado por mas de un índice, con lo que la rapidez de acceso se incrementa muchísimo.

Si indexamos un fichero de ventas, cuya clave es el número de factura, pero en su registro también guardamos la fecha, el cliente, el artículo, etcétera. Si ahora quisiéramos listar todos los registros de ventas

relativos a un solo cliente y no existieran las claves alternativas, tendríamos que leer todo el fichero para saber de que cliente es cada factura e imprimir solo esos registros. Con una clave auxiliar podemos posicionarnos directamente en el cliente en concreto y leer secuencialmente por esa clave, con lo cual tendremos una relación de todas sus facturas directamente hasta que el cliente cambie.

La programación en COBOL permite que creando una clave auxiliar, el ordenador no tenga que realizar dos tablas iguales pero indexadas cada una de una forma diferente, por lo que estas tablas no tendrán que ocupar el doble de espacio de disco.

Un ejemplo de fichero indexado con clave auxiliar será el siguiente:

```
SELECT FICHERO ASSIGN TO RANDOM 'FICHERO.DAT
ORGANIZATION INDEXED ACCESS DYNAMIC
RECORD KEY DNI
ALTERNATE RECORD KEY NOMBRE WITH DUPLICATES
FILE STATUS STA-FICHERO.
```

```
FD FICHERO LABEL RECORD STANDARD.
```

```
01 REG-FICHERO.
```

```
02 DNI PIC 9(5).
```

```
02 NOMBRE PIC X(30).
```

```
02 DOMICILIO PIC X(30).
```

```
02 POBLACION PIC X(20).
```

OPEN

OPEN (EXCLUSIVE) modo nombre de archivo (WITH LOCK) (WITH NO REWIND)

Se utiliza para abrir archivos. Estos archivos podrán ser de los siguientes tipos:

INPUT, el archivo se abrirá solo para lectura, es decir no podremos grabar ni modificar datos del mismo.

I-O, el archivo se abrirá como lectura y escritura, con lo cual tendremos acceso a toda la información de dicho archivo para leerla, escribirla, reescribirla o borrarla.

OUTPUT, el archivo se abre solo para escritura, es el formato que se utiliza en los archivos de impresión y secuenciales. Tiene la particularidad que crea el fichero nuevo cada vez que se utiliza, por lo tanto hay que tener cuidado con archivos Indexados.

EXTEND, igual que el anterior pero no crea el archivo, sino que la información se va añadiendo a la ya existente. Se utiliza para archivos secuenciales.

Los más utilizados son INPUT e I-O, ya que son los que funcionan de forma correcta con los archivos indexados que son los más utilizados en COBOL.

Las opciones EXCLUSIVE y WITH LOCK, nos indica cuando trabajamos en multipuesto que éste archivo estará bloqueado, es decir que no estará disponible para otros usuarios. El hecho de que existan dos opciones para lo mismo es por compatibilidad con versiones anteriores.

La opción WITH NO REWIND, se utiliza cuando utilizamos archivos de cinta, para que no la rebobine al abrirla.

CLOSE

Es la instrucción contraria a OPEN, y con la cual termina la conexión establecida con el archivo. A partir del momento que aparezca ésta instrucción el archivo no estará disponible para operar con él, hasta la próxima vez que se abra. Obviamente antes de cerrarlo debe de estar abierto.

CLOSE nombre de archivo (WITH LOCK) (WITH NO REWIND)

El nombre de archivo corresponderá a algún archivo abierto anteriormente.

Las opciones WITH LOCK y WITH NO REWIND , tienen la misma explicación que la vista en la orden OPEN.

READ

Es la instrucción que utilizamos para leer registros de un archivo que debe de estar abierto. Con ella conseguimos que los datos referentes al registro accedido queden en la descripción de dicho fichero, es decir, conseguimos que los campos declarados en la FD, tengan el valor correspondiente al registro leído.

La sentencia READ, se utiliza para leer ficheros de forma secuencial o indexada.

Formato para leer ficheros de manera secuencial. (Indexados o secuenciales)

READ nombre de archivo (NEXT/PREVIOUS RECORD) (INTO descripción) (AT END / NO AT END sentencia)

END-READ

nombre de archivo corresponderá a algún archivo abierto anteriormente.

La opción NEXT RECORD, indica que se va a leer el siguiente registro y es la que se toma por defecto, ya que, cuando estamos leyendo un archivo de forma secuencial, éste leerá registros uno tras otro, hasta llegar al final.

La opción PREVIOUS RECORD leería el registro anterior. Esta opción es la única que no es válida para ficheros secuenciales de éste formato.

La opción INTO, indica cual de las descripciones de registro que hayamos podido declarar será la que almacene los datos del registro leído. Tenemos que tener en cuenta que Cobol nos permite mantener mas de una descripción de registro para un mismo archivo. Si tuvieramos mas de una, ésta sería la opción para indicarle cual es la que queremos utilizar en ésta lectura.

La sentencia que va después de AT END , indica que debe de hacer el programa al llegar al final del fichero. Puede ser cualquier orden de cobol, pero es evidente que si volvemos a leer una vez llegado al final, producirá un error.

Formato para leer ficheros indexados con acceso aleatorio.

READ nombre de archivo (INTO descripción) (KEY nombre de clave) (INVALID KEY / NOT INVALID KEY sentencia)

END-READ

Las opciones que se repiten con el formato anterior tienen el mismo formato y producen el mismo resultado.

La opción KEY, indica por que clave se va a leer el fichero, siempre que éste tenga más de una.

La sentencia después de INVALID KEY se utiliza para ejecutar una acción cuando se intenta acceder a un registro que no existe. En el caso de utilizar NOT INVALID KEY sería al contrario, es decir cuando el registro existe.

WRITE

Con ésta instrucción se consigue grabar la información contenida en ese momento en los campos del registro de un fichero. Es decir, si introducimos una ficha nueva en la agenda con los datos de un nuevo amigo, ésta instrucción será la que nos sirva para almacenar en el fichero los datos. A partir de ese momento

estarán disponibles tantas veces como queramos para leerla. Y por supuesto el fichero debe de estar abierto como OUTPUT, EXTEND o I-O.

```
WRITE nombre de registro ( FROM descripción ) ( INVALID KEY / NOT INVALID  
KEY sentencia )  
END-WRITE
```

La opción FROM, indica con cual de las descripciones de registro que hayamos podido declarar se graben los datos en el fichero. Hay que señalar que ésta descripción puede estar definida en la WORKING, y lo que nos ahorra en realidad es mover los datos de esa descripción que hemos usado como "temporal" a la auténtica descripción del registro.

Las cláusulas de INVALID KEY y NOT INVALID KEY, tienen la misma función dada en la instrucción READ. Solo que aquí, INVALID KEY, se produciría cuando al grabar el registro, éste ya existiese o hubiera algún error por el cual no se pudieran grabar los datos.

Si lo que queremos es imprimir, también usaremos la cláusula WRITE, pero de una forma algo diferente a la anterior, añadiendo AFTER y PAGE.

```
WRITE nombre de registro ( FROM descripción ) ( AFTER número de  
líneas )(PAGE)  
END-WRITE
```

Además para éste formato tendremos la cláusula AFTER, en la cual indicamos el número de líneas que debe de avanzar la impresora antes de escribir, o bien que lo haga directamente al principio de la siguiente página, poniendo AFTER PAGE.

También estaría bien recordar que habrá que introducir el siguiente código:

```
FILE-CONTROL  
SELECT IMPRESORA ASSIGN TO PRINT "LPT1".  
...  
PROCEDURE DIVISION.  
...  
OPEN OUTPUT IMPRESORA
```

REWRITE

Esta instrucción se utiliza para volver a grabar datos de un registro ya existente. Toda la sintaxis es exactamente igual que la explicada en WRITE, por lo que no hace falta repetirla.

DELETE

Es una instrucción que permite borrar registros de un fichero. Su sintaxis es la siguiente:

DELETE nombre de fichero (INVALID KEY / NOT INVALID KEY sentencia)
END-DELETE

Las únicas cláusulas INVALID y NOT INVALID KEY, se usan exactamente igual que en las demás relativas a ficheros, es decir ejecutará la sentencia que pongamos a continuación cuando una de las condiciones se cumpla, que la clave exista o que no exista.

Si nos fijamos vemos que la gran diferencia está en que aquí la orden hace referencia al nombre del fichero y no al del registro, como en las instrucciones anteriores.

START

Es la instrucción que nos permite posicionarnos en cualquier parte del fichero, para una lectura mas rápida. Si imaginamos un fichero con 10.000 clientes, clasificados por código, para ver todos los que cuyo código es mayor a 9.000, tendríamos que leernos el fichero secuencialmente hasta llegar al sitio correcto, en cambio con esta orden, podremos colocarnos en la posición del fichero que queramos dentro de unas normas, que veremos a continuación.

START nombre de fichero KEY (expresión) nombre de clave (INVALID KEY / NOT INVALID KEY sentencia)
END-START

Las únicas cláusulas INVALID y NOT INVALID KEY, se usan exactamente igual que en las demás relativas a ficheros, es decir ejecutará la sentencia que pongamos a continuación cuando una de las condiciones se cumpla, que la clave exista o que no exista.

Si nos fijamos vemos que la gran diferencia está en que aquí la orden hace referencia al nombre del fichero y no al del registro, como en las instrucciones anteriores.

La expresión a la que se hace referencia en la sintaxis, pueden ser las siguientes:

LESS (<) menor que.

NOT LESS (NOT <) no menor que.

EQUAL (=) igual a.

GREATER (>) mayor que.

NOT GREATER (NOT >) no mayor que.

GREATER OR EQUAL (>=) mayor o igual que.

LESS OR EQUAL (<=) menor o igual que.

FIRST principio de fichero (RM/COBOL).

LAST final del fichero (RM/COBOL).

Ejemplo completo de programa con ficheros:

IDENTIFICATION DIVISION.

PROGRAM-ID. TRASPASO.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SPECIAL-NAMES. DECIMAL-POINT IS COMMA.

INPUT-OUTPUT SECTION.

FILE-CONTROL

SELECT CLIENTES ASSIGN TO RANDOM "CLIENTES.DAT"

- ORGANIZATION INDEXED ACCESS DYNAMIC RECORD CLIA-CLI

- FILE STATUS STA-CL

SELECT TRASPASO ASSIGN TO RANDOM ELNOMBRE

- ORGANIZATION LINE SEQUENTIAL.

DATA DIVISION.

FILE SECTION.

FD CLIENTES LABEL RECORD STANDARD.

01 REG-CLIENTES.

02 CLI-CODIGO PIC 9(6).

02 CLI-NOMBRE PIC X(30).

02 CLI-DOMICILIO PIC X(30).

02 CLI-CPOSTAL PIC 9(5).

02 CLI-POBLACION PIC X(30).

02 CLI-PROVINCIA PIC X(20).

FD TRASPASO LABEL RECORD STANDARD.

01 REG-TRASPASO.

02 TRA-CODIGO PIC 9(6).

02 TRA-SEP1 PIC X.

02 TRA-NOMBRE PIC X(30).

02 TRA-SEP2 PIC X.

02 TRA-DOMICILIO PIC X(30).
02 TRA-SEP3 PIC X.
02 TRA-CPOSTAL PIC 9(5).
02 TRA-SEP4 PIC X.
02 TRA-POBLACION PIC X(30).
02 TRA-SEP5 PIC X.
02 TRA-PROVINCIA PIC X(20).

WORKING-STORAGE SECTION.

01 TABLAMES.
02 FILLER PIC X(30) VALUE 'ENEFEBMARABRMAYJUNJULAGOSEPOCT'.
02 FILLER PIC X(6) VALUE 'NOVDIC'.
01 TABLIMES REDEFINES TABLAMES.
02 ELEMES PIC XXX OCCURS 12 TIMES.
01 FECHA PIC 9(6).
01 FECHO REDEFINES FECHA.
02 FANIO PIC 99.
02 FMES PIC 99.
02 FDIA PIC 99.
01 NO MBREAR.
02 NOM-MES PIC XXX.
02 NOM-ANIO PIC 99.
02 NOM-SEP PIC X.
02 NOM-DIA PIC 99.
02 NOM-EXT PIC XXXX.
01 FINFIC PIC X.

PROCEDURE DIVISION.

INICIO.

ACCEPT FECHA FROM DATE

MOVE ELEMES (FMES) TO NOM-MES

MOVE FANIO TO NOM-ANIO

MOVE FDIA TO NOM-DIA

MOVE '-' TO NOM-SEP MOVE '.TXT' TO NOM-EXT
MOVE NO MBREAR TO ELNOMBRE

OPEN INPUT CLIENTES OUTPUT TRASPASO

MOVE ' ' TO FINFIC

PERFORM UNTIL FINFIC = 'S'

READ CLIENTES NEXT RECORD AT END MOVE 'S' TO FINFIC

NOT AT END

MOVE CLI-CODIGO TO TRA-CODIGO

MOVE CLI-NOMBRE TO TRA-NOMBRE

MOVE CLI-DOMICILIO TO TRA-DOMICILIO

MOVE CLI-CPOSTAL TO TRA-CPOSTAL

MOVE CLI-POBLACION TO TRA-POBLACION

MOVE CLI-PROVINCIA TO TRA-PROVINCIA

MOVE 'I' TO TRA-SEP1 TRA-SEP2 TRA-SEP3 TRA-SEP4 TRA-SEP5

WRITE REG-TRASPASO

END-READ

END-PERFORM

CLOSE TRASPASO CLIENTES

STOP RUN.

Códigos de error.

Son muchos los errores que se pueden producir en tiempo de ejecución cuando trabajamos con archivos, a continuación daré una explicación de los mas comunes. Todos éstos errores los podemos obtener si definimos FILE STATUS en la FILE-CONTROL, y actuar consecuentemente, por ejemplo yo siempre cuando abro los ficheros al principio de cada programa compruebo que el error sea 00, es decir que todo está bien para continuar, sino automáticamente saco una ventanita con el error que se ha producido y el nombre del fichero para que mis usuarios me lo comuniquen y poder solucionarlo, pero no dejo que se siga ejecutando el programa, con lo que se evitarán problemas posteriores que podrían resultar peores. De todas formas una vez que está el programa verificado y comprobado, y que todos los archivos se corresponden con su descripción en la FD, los errores que nos puede dar son los relativos a claves duplicadas o inexistentes, o a errores del sistema.

Los errores se representan como 2 dígitos para el error que pueden ir seguidos de una coma y otros 2 dígitos para indicar la naturaleza del error.

ERROR	EXPLICACION
00	Operación satisfactoria, ausencia de errores.
10	Cuando se ha llegado al final del fichero y se quiere seguir leyendo.
22	Se intenta copiar un registro con una clave ya existente.
23	Cuando se hace un acceso directo a un registro inexistente.
24	No hay espacio en disco para realizar la operación.
30	Es un error grave de entrada/salida, suele ser ajeno a Cobol y más concreto en cuanto a configuración del sistema operativo sobre el que se está ejecutando, para solucionarlo deberemos prestar atención a los dos dígitos siguientes al error.
34	Igual que el 24 por falta de espacio en disco.
35	El archivo al que hace referencia no existe.
39	La organización del fichero que se quiere abrir no coincide con su organización real interna. Suele pasar cuando se modifica una FD y el fichero continúa siendo el mismo.
41	Cuando intentamos abrir un archivo que ya está abierto.
42	Si intentamos cerrar un archivo que no esta abierto.
43	Cuando se quiere borrar o re-escribir un registro en un fichero abierto con acceso secuencial.

47	Se quiere realizar alguna operación que no corresponde con el modo de acceder al fichero. Si queremos hacer un READ o un START en un archivo que no ha sido abierto como INPUT o como I-O.
48	Igual que el anterior, pero para el caso de querer escribir en un fichero que no esta abierto o no está abierto como I-O, OUTPUT o EXTEND.
49	Cuando queremos borrar o re-escribir en un fichero que no ha sido abierto como I-O o el fichero no está abierto.
90	Cuando COBOL recibe un mensaje irreconocible, suele estar ocasionado por incompatibilidad entre los accesos a los ficheros, su forma de apertura y su modo de acceso.
91	Igual que el 42.
92	Cuando intentamos leer, escribir, borrar, re-escribir sobre un archivo no abierto.
93	El archivo está en conflicto con otro usuario.
94	Igual que el 39, tiene muchísimas vertientes dependiendo de los dos dígitos siguientes que acompañan al error.
98	<p>Échate a temblar (perdonad por la expresión), grave error interno en la estructura del fichero, generalmente se suelen provocar cuando hay un corte eléctrico o una salida brusca del programa y algún fichero indexado se ha quedado abierto, lo que suele ocurrir es problema de correspondencia entre claves y datos, se recomienda hacerle un RECOVERY (RM/COBOL) o el que corresponda a cada compilador para rehacer las claves. Si después de todo sigue dando errores?</p> <p>En algunos casos, es posible que éste error se deba a un problema físico del disco, en ese caso también será conveniente copiarlo en otro sitio "físico".</p>
99	Al borrar, leer o re-escribir un registro que está siendo bloqueado por otro usuario.

Sentencia SORT.

La clasificación en los ficheros es muy importante. Por lo general se suelen utilizar ficheros indexados por clave principal y clave auxiliares, pero si el archivo no está indexado por un campo en concreto, la sentencia SORT es la que nos permitirá ordenar el fichero por ese archivo.

Podemos clasificar por todos los campos que deseemos, así como escoger el orden que queramos para cada uno de ellos. Todo esto queda englobado en la sentencia SORT, pero esta depende de muchas mas cosas.

Lo primero que necesitamos es un archivo sobre el que volcar nuestra clasificación y que además utilizará el propio SORT para hacer las clasificaciones. Este fichero se define en la FILE-CONTROL como cualquier otro fichero solo que haciendo ver al compilador que se va a tratar de un archivo SORT y a continuación se define su estructura en la FILE SECTION.

Para ver como funciona la sentencia, utilizaremos el siguiente ejemplo para poder entender la sentencia mejor.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL

- SELECT AGENDA ASSIGN TO RANDOM "AGENDA.DAT"
- ORGANIZATION INDEXED ACCESS DYNAMIC
- RECORD KEY KEYAGE
- SELECT ORDEN ASSIGN TO SORT

DATA DIVISION.

FILE SECTION.

FD AGENDA LABEL RECORD STANDARD.

01 REGAGE

02 KEYAGE

03 AGE COD PIC 9(4).

02 AG ENOM PIC X(30).

02 AG EDOM PIC X(30).

02 AGE POS PIC 9(5).

02 AGE POB PIC X(20).

02 AGE PRO PIC X(15).

```

      02 AGETEL      PIC X(20).
      02 AGEMOV      PIC X(20).
      02 AGEMAI      PIC X(30).
      02 AGEWEB      PIC X(40).
SD  ORDEN.
01  RECORD.
      02 ORDNO M     PIC X(30).
      02 ORDPO B     PIC X(30).
      02 ORDPRO      PIC X(30).

```

En lugar de escribir FD hemos usado SD (SORT Description)

Hay dos formas distintas de ordenar los ficheros, la primera de las formas es la que clasifica los datos en un fichero segundo, usando un tercer fichero, su formato es el siguiente:

```

SORT fichero_de_clasificación
  (ON ASCENDING KEY) campo
  (ON DESCENDING KEY) campo
USING fichero_a_clasificar
GIVING fichero_que_quedará_clasificado

```

La segunda forma ordena directamente el fichero, sin utilizar ningún otro:

```

SORT fichero_de_clasificación
  (ON ASCENDING KEY) campo
  (ON DESCENDING KEY) campo
INPUTPROCEDURE qué_hacemos_antes
OUTPUTPROCEDURE qué_hacemos_después

```

Fichero_de_clasificación: hace referencia al fichero que hemos definido como SORT.

campo: hace referencia a cualquiera de los campos de dicho fichero que hemos definido, podremos poner tantos como deseemos y en el orden que queramos.

qué_hacemos_antes: indica el proceso que obligaremos al compilador a realizar antes de clasificar.

Podemos indicar varios párrafos con THRU o bien utilizar una SECTION.

qué_hacemos_después: indicaremos que hacer una vez el fichero esté clasificado.

PROGRAMACIÓN EN COBOL	1
Estructura del lenguaje COBOL	1
Juego de caracteres.....	1
Palabras en COBOL:.....	2
Palabras reservadas de COBOL, variables, símbolos y constantes.....	2
Estructura de los programas COBOL.....	6
Organización de los datos.....	6
Declaraciones, sentencias, párrafos, y divisiones.....	6
Programa Fuente.....	6
Divisiones de un programa COBOL.....	7
IDENTIFICATION DIVISION.....	8
PROGRAM ID.....	8
AUTHOR.....	8
INSTALLATION.....	8
DATE-WRITTEN.....	8
DATE-COMPILED.....	8
REMARKS.....	8
ENVIRONMENT DIVISION.....	10
CONFIGURATION SECTION.....	10
SOURCE-COMPUTER.....	10
OBJECT- COMPUTER.....	10
ESPECIAL NAMES.....	10
INPUT-OUTPUT SECTION.....	11
FILE CONTROL.....	11
SELECT.....	11
ASSIGN TO.....	12
ORGANIZATION IS.....	12
ACCESS MODE IS.....	12
RECORD KEY.....	12
ALTERNATE RECORD KEY.....	13
FILE STATUS.....	13
I-O CONTROL.....	13
DATA DIVISION.....	14
FILE SECTION.....	14

WORKING STORAGE SECTION	15
LINKAGE SECTION.....	15
SCREEN SECTION.	19
PROCEDURE DIVISION.....	20
Sentencias básicas.	22
ACCEPT	22
DISPLAY	23
COPY	25
MOVE	26
INITIALIZE	26
INSPECT	27
STRING	29
UNSTRING	31
Sentencias de control.....	33
IF ... THEN ... ELSE	33
PERFORM	33
STOP	33
Instrucciones Matemáticas.....	34
ADD	34
SUBTRACT	35
MULTIPLY	35
DIVIDE	36
COMPUTE	36
Manipulación de tablas.....	37
Ficheros de entrada y salida	39
OPEN	40
CLOSE	41
READ	41
WRITE	42
REWRITE	44
DELETE	44
START	44
Códigos de error.	49
Sentencia SORT.	51