

# CURSO COBOL



## ACMSAP – ERP LEARNING

Juan Martínez Villergas, 2, 1º D

47014 – Valladolid

Teléfono:

[www.acmsap.es.tl](http://www.acmsap.es.tl)

acmsap@yahoo.es

## NOTAS PRELIMINARES

La informática, como ciencia y técnica de la segunda mitad del siglo XX, está en continuo cambio, la investigación y el desarrollo tecnológico amplían constantemente sus posibilidades, de modo especial en todo lo relativo a los lenguajes de programación. Por esta razón, hemos procurado que las especificaciones del COBOL presentadas en este manual correspondan al estado del mismo en el momento de concluir.

El cobol es un lenguaje industrial, y no es propiedad de ninguna empresa o grupo de empresas, ni tampoco de ninguna organización o grupo de organizaciones.

El COBOL es un lenguaje artificial, similar en muchos aspectos al lenguaje hablado en inglés, que permite al hombre comunicarse con el ordenador. El vocablo COBOL es una contracción de la frase.

### COmmon Business Oriented Language

El primer diseño del COBOL se debe a un consenso entre las administraciones públicas de USA, los fabricantes de ordenadores, las universidades y las organizaciones de los usuarios.

Las aplicaciones comerciales difieren sustancialmente de las científicas en la cantidad de datos a tratar y en la complejidad de los cálculos. En general, éstas últimas requieren mucho más cálculo y poca entrada y salida de datos sin previo diseño de formato, mientras que los problemas de gestión precisan manipular gran cantidad de datos, cuyos formatos están ya establecidos, a los que se les suele aplicar operaciones aritméticas sencillas. Normalmente, este tipo de aplicaciones implica el tratamiento de ficheros de datos de entrada y salida.

Por todo ello, los objetivos exigidos al nuevo lenguaje fueron los siguientes:

- sintaxis cercana al lenguaje hablado; naturalmente en inglés.
- Uso restringido de símbolos especiales.
- Máxima potencia en el tratamiento de ficheros.
- Instrucciones de cálculo reducidas al mínimo imprescindible para la gestión administrativa.
- Amplias posibilidades de evolución futura.
- Independencia del ordenador empleado.

A lo largo del tiempo en que se ha utilizado, el COBOL ha sufrido mejoras y ampliaciones, de modo que no sólo se dispone de diferentes versiones para ordenadores de distintos fabricantes, sino también para un mismo modelo de ordenador. Sin embargo, las variaciones de una versión a otra suelen ser pequeñas.

En el curso se aprenderá el COBOL ANS-85. Publicación de la norma ANSI X3.23-1985. Lenguaje de programación COBOL.

## 1. DESCRIPCION DEL PROGRAMA.

Un programa escrito en cualquier Lenguaje necesita como mínimo 3 elementos fundamentales:

- Un conjunto de DATOS
- Un conjunto de DEFINICIONES (Entradas / Salidas).
- Un conjunto de INSTRUCCIONES (Sentencias).

### UN PROGRAMA ESCRITO EN LENGUAJE COBOL SE ESTRUCTURA DE LA FORMA SIGUIENTE:

- **Cuatro grandes áreas**, llamadas “DIVISIONES”, dentro de cada una de las cuales, se aporta distintas informaciones. Estas informaciones y divisiones son:
- Para la ASIGNACIÓN de nombre al programa e identificar al programador, se realiza en:

#### \* La IDENTIFICATION DIVISION

- Para el ENLACE del programa con los recursos físicos donde residirán los datos, se codifica en:

#### \* La ENVIRONMENT DIVISION

- Para definir el MAPA o DICCIONARIO DE DATOS que han de ser procesados, así como las áreas específicas de trabajo, se utiliza:

#### \* La DATA DIVISION

- Para codificar PROCEDIMIENTOS DE PROCESO, es decir, para escribir las instrucciones o mandatos para manipulación y procesamiento de los datos, se emplea:

#### \* La PROCEDURE DIVISION

En cada una de las DIVISIONES, a su vez, se pueden desglosar sus funciones, agrupándolas, por SECCIONES y/o PÁRRAFOS.

Dentro de cada SECCIÓN o PÁRRAFO se escriben CLAÚSULAS o SENTENCIAS.

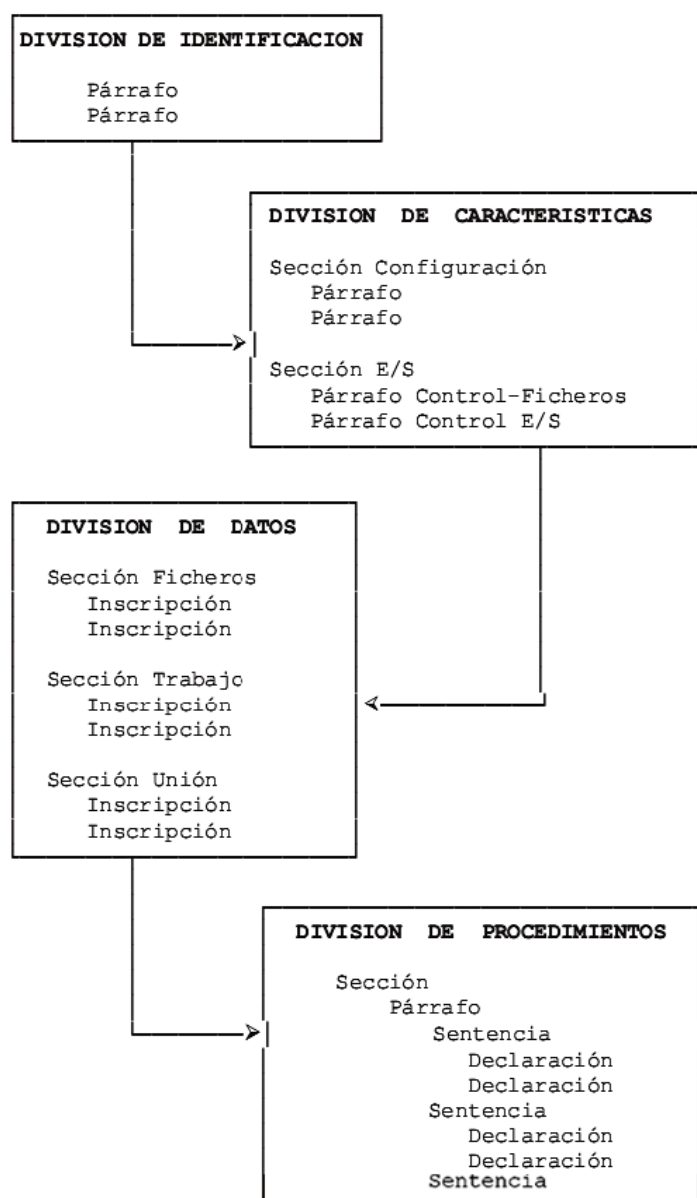
**CLASULA:**

Sirve para especificar atributos y características de campos y de archivos y se utiliza en: La IDENTIFICATION, La ENVIRON MENT y en La DATA DIVISION.

**SENTENCIA:**

Determina la ejecución de algún tipo de acción y se utiliza en:

La PROCEDURE DIVISION.

**2. ESQUEMA GENERAL DEL PROGRAMA**

### 3. ELEMENTOS DEL LENGUAJE

El COBOL utiliza en su sintaxis un juego reducido de PALABRAS inglesas además de los símbolos aritméticos convencionales.

El **carácter** es la partícula más elemental del lenguaje.

La correcta combinación de caracteres forman las PALABRAS (Instrucciones o Sentencias) que constituyen las órdenes que controlan al ordenador.

Cualquier elemento del lenguaje COBOL estará formado por conjuntos de los siguientes caracteres:

Caracteres Alfabéticos: Letras desde la **A** a la **Z** y el "**Blanco o SPACE**".

Caracteres Numéricos: Números desde el **0** al **9**.

Caracteres empleados en operaciones aritméticas: **( ) , « » , « , / , + , - , =**

Caracteres Especiales empleados como operadores relacionales: **> , < , =**

#### 4. FUNCIONES ESPECIALES DE LOS CARACTERES

CARACTER	FUNCION
<b>b    blanco</b>	Separador de elementos.
<b>.    punto</b>	Edición y puntuación.
<b>&lt;   menor que</b>	Operador relacional.
<b>&gt;   mayor que</b>	Operador relacional.
<b>=   igual a</b>	Operador relacional.
<b>()   paréntesis</b>	Como aisladores de modelos y en expresiones y subíndices.
<b>+   signo más</b>	Como operador aritmético y en modelos de edición.
<b>-   signo menos</b>	Como operador aritmético en modelos de edición y para formar palabras.
<b>*   asterisco</b>	Como carácter de edición, en palabras COBOL, operador aritmético y para comentario.
<b>/   barra</b>	Como operador aritmético y como carácter de edición.
<b>,   coma</b>	Como carácter de puntuación y para edición.
<b>\$   signo dólar</b>	Para edición.
<b>' o "   comilla/s</b>	Aisladores de literales.
<b>A a Z</b>	Construcción de palabras COBOL.
<b>0 a 9</b>	Construcción de cifras y palabras COBOL.

## 5. PALABRAS COBOL

Se entiende por palabras COBOL al conjunto de no más de 30 caracteres formado de acuerdo con unas reglas predefinidas, con las cuales se codifica el programa.

Las palabras COBOL pueden ser CREADAS o RESERVADAS.

### 5.1 PALABRAS COBOL CREADAS

Se trata de palabras que el programador inventa para identificar campos, registros, ficheros, programas, etc.

Para construir estas palabras habrá que tener en cuenta las normas siguientes:

Nombre de	<ul style="list-style-type: none"><li>- CAMPO</li><li>- CONDICION</li><li>- REGISTRO</li><li>- FICHERO</li></ul>	De 1 a 30 caracteres, el primero deberá ser letra. Sin signos especiales, ni espacios, excepto el guión (-), el cual no irá ni al principio ni al final.
Nombre de	<ul style="list-style-type: none"><li>- PARRAFO</li></ul>	Todo exactamente igual que los nombres de arriba, excepto que los párrafos pueden comenzar por números.
Nombre de	<ul style="list-style-type: none"><li>- PROGRAMA</li></ul>	De 1 a 8 caracteres. Sin signo especial alguno, ni blancos y el primero, al menos, será una letra. (Se podrían codificar hasta 30 caracteres, pero si se ponen más de 8, se truncan).

### 5.2 PALABRAS COBOL RESERVADAS

Se trata de palabras con **sentido fijo** en el lenguaje COBOL. Por tanto, serán utilizables solamente dentro de un entorno determinado. A continuación se detallan las palabras reservadas en COBOL II.



## 6. PLANTILLA DE CODIFICACIÓN DEL PROGRAMA COBOL

La plantilla para la codificación del programa COBOL consta de 80 columnas distribuidas de la siguiente forma:

### **1 a 6 : Área de secuencia**

Seis dígitos numéricos que identifican cada una de las líneas de programa. Es la numeración COBOL.

### **7 : Área indicativa de continuación o comentario**

La presencia de un guión (-) en esta línea indica que en la misma continua un literal iniciado en la línea anterior.

La presencia de un asterisco («) indica que se trata de una línea de comentario.

### **8 a 11 : Área A**

Área en la que habrá que iniciar la codificación de algunos elementos COBOL, como se indicará más adelante.

### **12 a 72 : Área B**

Área en la que habrá que iniciar la codificación de algunos elementos COBOL, como se indicará más adelante.

### **73 a 80 :**

Área utilizada, opcionalmente para codificar el nombre del programa. (También puede quedar en blanco o solicitar la numeración Standard).

PAG	PROGRAMA				SISTEMA								HOJA N°								DE			
1 3	PROGRAMADOR				FECHA								IDENTIFICACION 73								80			
LIN	A		B																					
4 6	7	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72						
001																								
002																								
003																								
004																								
005																								
006																								
007																								
008																								
009																								
010																								
011																								
012																								
013																								
014																								
015																								
016																								
017																								
018																								
019																								
020																								

## 7. NORMAS PARA INTERPRETAR LOS FORMATOS

- LAS **MAYÚSCULAS** son palabras **reservadas** del COBOL.
- Si la palabra reservada aparece **SUBRAYADA** en el formato, su codificación es obligatoria.
- Los **nombres de variables** y literales aparecerán en MINUSCULAS.
- El uso de las palabras entre corchetes “[ ]” es OPCIONAL.
- Cuando es preciso seleccionar una, entre varias opciones, estas aparecerán entre llaves “{ }”.
- Los puntos suspensivos (...) indican que algunas opciones pueden repetirse varias veces.

## **8. GENERALIDADES SOBRE COMPILACIÓN, ENSAMBLE Y PUESTA A PUNTO DE PROGRAMAS.**

Cuando se decide crear un programa, hasta que éste queda completamente depurado y se pueda dejar disponible para que el departamento de explotación lo “ejecute” cuando considere necesario, hay que realizar normalmente los siguientes pasos:

- « COMPILACION DEL PROGRAMA FUENTE.
- « ENSAMBLE, o ENLACE, o “LINKEDITACION”.
- « EJECUCION DEL PROGRAMA.
- « ANALISIS DE LOS RESULTADOS GENERADOS.
- « Y SI FUERA NECESARIO, CORREGIR EL PROGRAMA Y VOLVER AL PASO PRIMERO.

## **9. COMPILACIÓN DEL PROGRAMA COBOL.**

Un programa codificado en un lenguaje simbólico se escribirá en hojas COBOL o se grabará directamente en un soporte magnético (disco).

Las INSTRUCCIONES de un programa en esta situación no pueden “ejecutarse”, porque la UCP no reconoce el lenguaje en que están escritas.

Por tanto, es imprescindible que cada una de las instrucciones escritas en lenguaje simbólico sean ‘traducidas’ a un lenguaje que la UCP reconozca.

La función de TRADUCIR instrucciones desde el lenguaje simbólico al lenguaje inteligible al ordenador la realizan unos programas especiales, denominados programas COMPILADORES.

En nuestro caso, la función la realizará el compilador de COBOL.

## **DEFINICIÓN DE CONCEPTOS**

**PROGRAMA FUENTE:** Programa cuyas instrucciones están escritas en un lenguaje simbólico.

**PROGRAMA OBJETO:** Programa cuyas instrucciones están escritas en lenguaje reconocible por el ordenador.

**COMPILAR:** Fundamentalmente acción de traducir un programa FUENTE a OBJETO.

**COMPILADOR:** programa que realiza básicamente la función de traducir un program FUENTE a OBJETO.

**OTRAS FUNCIONES DEL COMPILADOR:**

Al realizar una compilación se le puede pedir al compilador que realice diversas funciones.

Una de las funciones que realiza dicho parámetro es la de producir un LISTADO DEL PROGRAMA FUENTE, como se indica anteriormente.

Otra de las funciones es la DEPURACION SINTÁCTICA del programa FUENTE. El compilador 'detecta' los posibles errores sintácticos, cometidos al codificar el programa, y produce un listado indicando la línea donde se ha detectado y la regla sintáctica que se ha infringido.

## **10. LINKEDITACION, ENSAMBLE O ENLACE DE PROGRAMAS.**

Un programa FUENTE que ha sido compilado y por tanto traducido a lenguaje máquina, es decir, convertido en programa OBJETO, es un programa que todavía no es EJECUTABLE.

No es EJECUTABLE, porque está incompleto.

Cada una de las instrucciones de un programa FUENTE, al ser 'traducido' a OBJETO, se 'EXPANDEN' en varias INSTRUCCIONES MÁQUINA. Estas instrucciones, además, son ordenadas y direccionadas secuencialmente.

Existen, sin embargo, verbos COBOL que necesitan un número de instrucciones complejas para poderse EJECUTAR.

Estos conjuntos de INSTRUCCIONES MAQUINA se almacenan en unas LIBRERÍAS, formando MÓDULOS localizables mediante unos nombres que el compilador 'asocia' a determinados verbos COBOL.

Por tanto, en el programa OBJETO **no están** todavía las instrucciones de esos MODULOS, aunque estos sí están definidos y direccionados.

Además, existen instrucciones COBOL que piden la ayuda de otros programas. Por ejemplo, el verbo CALL.

El compilador direcciona la instrucción, pero no incorpora el programa LLAMADO. El programa "LINKEDITADOR", tiene como función básica incorporar (ENLAZAR), tanto los módulos COBOL definidos por el compilador, como los programas o subprogramas solicitados vía CALL.

Con el conjunto formado por el programa OBJETO, más los módulos COBOL, más los subprogramas añadidos, genera lo que se denomina programa EJECUTABLE.

### **11. EJECUCION Y PUESTA A PUNTO DE PROGRAMAS.**

Una vez que el programa ha sido LINKEDITADO, ya SÍ se puede ejecutar.

Los PROGRAMAS que de forma automática desarrollan los Sistemas de la Compañía (nóminas, control de producción, etc..) son, como es lógico, PROGRAMAS EJECUTABLES, que residen permanentemente en LIBRERÍAS.

Para que se ejecuten, basta con "llamarlos" o referenciarlos adecuadamente.

Sin embargo, hasta que un PROGRAMA pasa a SER EJECUTABLE, ha de recorrer un camino que podemos definir como de PUESTA a PUNTO.

Este camino a recorrer tiene como objetivo garantizar que el PROGRAMA **'funciona' y 'funciona bien'**. Sólo a partir de este momento el programa podrá pasar a residir en la librería de PROGRAMAS EJECUTABLES (programas en explotación).

Durante la PUESTA a PUNTO el programa se hará EJECUTABLE solamente el tiempo necesario para 'probarlo' y conseguir unos resultados. Si es necesario, se corregirán instrucciones y de nuevo se probará el Programa.

El circuito de la PUESTA a PUNTO será COMPILAR, LINKEDITAR y EJECUTAR tantas veces como se considere necesario.

Lógicamente en esta Fase intervienen también los DATOS de ENTRADA así como los RESULTADOS.

# DEFINICIÓN DE DATOS

## 1. INTRODUCCION A LA DEFINICION DE CAMPOS Y DATOS

Con independencia de la SECCION de la DATA DIVISION en que se definan, para cada uno de los datos que han de ser procesados, habrá que identificar:

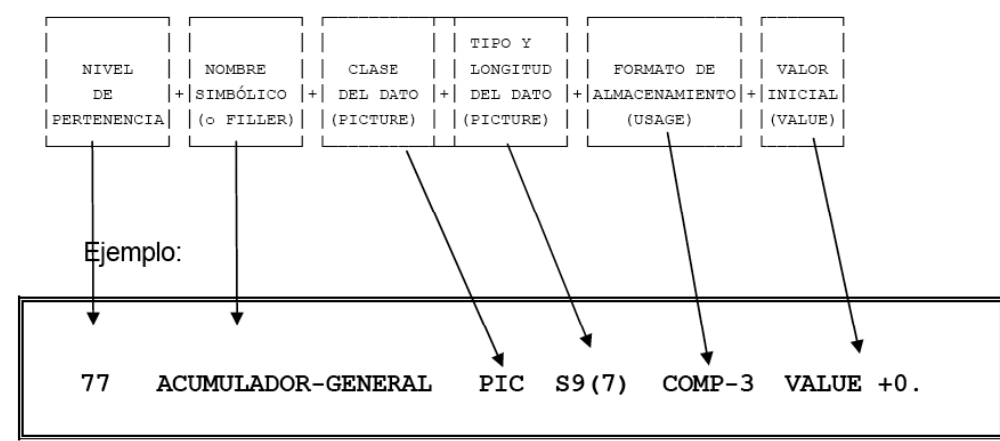
**LUGAR DE ALOJAMIENTO:** Nombre, longitud del campo.

**CARACTERÍSTICAS:** Clase, formato y uso del dato.

**PERTENENCIA:** A qué familia de datos pertenece.

Estas características se asignarán A CADA CAMPO en función de las necesidades particulares de cada proceso.

### FORMATO GENERAL DE DEFINICIÓN DE CAMPOS:



A continuación, se profundizará, en cada uno de estos conceptos.

## 2. IDENTIFICACIÓN DE DATOS. NOMBRES SIMBÓLICOS.

Los campos que han de soportar datos, se identifican mediante nombres simbólicos creados por el programador. Estos nombres se construyen de acuerdo con las normas indicadas en el párrafo siguiente.

### NORMAS DE CONSTRUCCION DE NOMBRES:

Conjuntos de 1 a 30 caracteres, el primero de los cuales será letra y sin signos especiales, excepto guiones internos.

CODI-PIEZA1	}	nombres válidos
C-O-D-I-G-O		
CODIGO-12		
CODIGO-	}	nombre inválidos
-NOMBRE		
CODIGO PIEZA		
CONTA LINEAS		
125BUCL		

**Nota:** Es recomendable definir nombres inteligibles y de acuerdo con algún tipo de norma predefinida.

## 3. DEFINICIÓN DE PERTENENCIA. NIVELES

Los datos en el programa están jerárquicamente organizados. En un nivel último los datos pertenecen a un FICHERO.

En COBOL se definen estas estructuras jerárquicas mediante la identificación de FICHEROS y DATOS por medio de NIVELES.

### 3.1 NIVELES ESPECÍFICOS Y NIVELES COMUNES

Existen **NIVELES PROPIOS o NIVELES ESPECÍFICOS** de algunas Secciones de la DATA DIVISION, como por ejemplo, los siguientes niveles:

**77:** Nivel que se emplea para la definición de campos simples, es decir, también conocidos como campos elementales. Estos, no forman parte de ningún otro campo. Este nivel se puede usar en la WORKING-STORAGE SECTION y en la LINKAGE SECTION.



**FD:** Nivel con el cual se inicia la descripción de un FICHERO en la FILE SECTION. (FD: File Description).

**SD:** Nivel que inicia la descripción de un FICHERO intermedio de trabajo en la FILE SECTION para clasificación interna de registros. (SD: Sort Description).

NOTA: Los niveles 77, FD y SD se comienzan a codificar en el margen A.

También existen **NIVELES COMUNES** a todas las SECCIONES.

**01:** Facilita normalmente la definición de un “campo compuesto”. Puede también definir un “campo elemental”.

**02 a 49:** Facilitan normalmente la definición de “campos elementales”. Pueden también definir un “campo compuesto”.

**88:** Define **nombres de condición**. Sus valores se asocian a la variable condicional de la que dependen.

**66:** Proporciona un **nuevo nombre** a un campo elemental, un campo compuesto o un grupo de campos elementales. Se asocia con la cláusula RENAME.

NOTA: Los niveles 01 y 66 se definen en el margen A. Los niveles 02 a 49 se definen en el margen B, y el 88 en los dos.

### 3.2 CAMPOS ELEMENTALES Y CAMPOS COMPUESTOS

En relación con los NIVELES, es necesario comentar con detenimiento, **una nueva división de los campos** en cuanto al concepto de **PERTENENCIA**.

Los campos se pueden subdividir en **ELEMENTALES** y **COMPUESTOS**.

#### **CAMPOS ELEMENTALES:**

Son campos **con entidad propia** procesables por sí mismos, sin que éstos incluyan a otros campos de menor entidad y longitud. Pueden pertenecer o no a otros campos jerárquicamente superiores e í.



```

010 | 01  REGISTRO-DE-VENTAS .
020 |     02  NUMERO-DE-VENTA      PIC  X(5) .
030 |     02  ZONA-VENTA .
040 |         03  PROVINCIA          PIC  X(2) .
050 |         03  AGENCIA            PIC  X(4) .
060 |         03  VENDEDOR           PIC  X(3) .
070 |
080 |     02  CLIENTE .
090 |         03  CODIGO-CLIENTE     PIC  9(4) .
100 |         03  RAZON-SOCIAL .
110 |             04  DIRECCION       PIC  X(30) .
120 |             04  TFNO .
130 |                 05  NUM-TFNO    PIC  9(7) .
140 |                 05  EXT-TFNO    PIC  9(3) .
150 |
160 |     02  ARTICULO .
170 |         03  CODIGO-ARTICULO     PIC  X(6) .
180 |         03  PRECIO-UNITARIO     PIC  S9(8)V9(2)  COMP-3 .
190 |         03  CANTIDAD            PIC  S9(5)      COMP-3 .
200 |         03  FECHA .
210 |             04  AÑO             PIC  X(4) .
220 |             04  MES             PIC  X(2) .
230 |             04  DIA             PIC  X(2) .
240 |
250 |

```

\* LA LONGITUD TOTAL DEL REGISTRO ES DE **80** BYTES .

### 5.3.3 NOMBRES SIMBÓLICOS DUPLICADOS Y CALIFICACIÓN

El nombre de los CAMPOS debe de ser **único** en un mismo programa. No obstante el COBOL permite que haya nombres duplicados, siempre y cuando estos nombres iguales (de campos de igual o distinto tipo y longitud) tengan otros nombres de datos **a nivel superior** que sean **distintos**.

LA TÉCNICA que se usa para distinguir campos con nombres iguales se llama **CALIFICACIÓN**.

Para **referenciar** un campo con nombre duplicado en la PROCEDURE DIVISION, habrá que indicar además de éste, el nombre del grupo inmediato a que pertenece el dato nombrado. Para ello se usa **OF** ó **IN** indistintamente.

Ejemplo:

```

01  REG-E.
    02  NOMBRE-E      PIC  X(12) .
    02  EDAD-E        PIC  9(3) .

01  REG-S.
    02  NOMBRE-S      PIC  X(12) .
    02  EDAD-S        PIC  9(3) .

```

.....

```

MOVE NOMBRE-E      TO NOMBRE-S .

```

```

MOVE EDAD-E        TO EDAD-S .

```

#### 4. PICTURE Y CLASE DE DATO

Se define la clase de un dato dependiendo del contenido del mismo. Esto se indica mediante la cláusula **PICTURE** (**PIC** abreviado).

- \* Será de clase **A** (**alfabético**) Si contiene sólo letras y/o blancos.
- \* Será de clase **X** (**alfanumérico**) Si contiene letras y/o dígitos y/o signos especiales.
- \* Será de clase **9** (**numérico**)

Si contiene sólo dígitos y signo válido.

**Ejemplos:**

**02 CODIGO PIC A(5).**

**77 CODIGO1 PIC X(7).**

**01 IMPORTE PIC 9(3).**

MASCARAS DE EDICION

Tipifican una clase de dato especial: DATO EDITADO, que por su complejidad se tratará aparte.

## 5. PICTURE Y LONGITUD DEL DATO

Se entiende por longitud el **número de caracteres** que van a ser alojados en el campo.

Esto se indica mediante la cláusula **PICTURE (PIC)** y el **Nº dígitos** entre paréntesis, indicando el número total de caracteres que soportará el campo.

También se puede dar la longitud de un campo, sin necesidad de dar el número de caracteres entre paréntesis; para ello bastaría con repetir la clase del campo (A, X o 9) una vez por cada uno de los caracteres que se desea almacenar en dicho campo.

Veamos los siguientes ejemplos:

Ejemplos:			
05	VALOR-X	PIC	X(15) .
05	VALOR-X	PIC	XXXXXXXXXXXXXXXXXX .
02	IMPORTE	PIC	9(8) .
02	IMPORTE	PIC	99999999 .

\* Lo más usado es: PIC X; PIC XX PIC x(N) siendo N >=3.

\* En Cobol la mínima unidad de memoria direccionada es el octeto, por tanto la menor PIC será:

PIC X o PIC A o PIC 9.

\* La cláusula PICTURE **sólo** se codifica en **campos elementales**.

\* La codificación de la "PICTURE" de cualquier campo, no podrá estar formada por más de 30

caracteres, con independencia de la longitud del campo que se esté definiendo .

\* En la definición de campos numéricos, el compilador, asignará un número mayor, menor o igual de BYTES DE MEMORIA para almacenar el número de dígitos que indica la longitud del campo, en función del formato de almacenamiento que se asigne a los datos.

\* Para campos numéricos con formato de coma fija, restringen la longitud hasta un máximo de 18 dígitos decimales. (Los de coma flotante no los consideraremos)

## 6. FORMATO DE LOS DATOS

Existen distintos formatos de almacenamiento de datos. Estos formatos son: FORMATO ZONA, FORMATO EMPAQUETADO y FORMATO BINARIO PURO. Para especificar el contenido de los campos vamos a usar el código EBCDIC.

### 6.1 FORMATO ZONA (VALORES NUMÉRICOS Y ALFANUMÉRICOS)

A razón de 1 carácter por octeto. (Cada uno de los caracteres del código EBCDIC ó ASCII se almacenan en un BYTE de memoria).

la letra <b>A</b>	<table><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td colspan="4">C</td><td colspan="4">1</td></tr></table>	1	1	0	0	0	0	0	1	C				1			
1	1	0	0	0	0	0	1										
C				1													
el dígito <b>5</b>	<table><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td colspan="4">F</td><td colspan="4">5</td></tr></table>	1	1	1	1	0	1	0	1	F				5			
1	1	1	1	0	1	0	1										
F				5													

#### \* Campos numéricos en formato de zona

El signo +, - o **absoluto** se localiza en los 4 primeros bits del último octeto de la cifra, de esta forma:

Signo +	=	<b>C</b> 1100
Signo -	=	<b>D</b> 1101
Signo <b>absoluto</b>	=	<b>F</b> 1111

Ejemplo:

La cifra <b>159</b>			signo ↓
159+	POSITIVO (+)	F1   F5   C9	
159-	NEGATIVO (-)	F1   F5   D9	
159	ABSOLUTO (+)	F1   F5   F9	

## 6.2 FORMATO EMPAQUETADO (SÓLO VALORES NUMÉRICOS)

A razón de 2 dígitos por octeto, excepto el octeto extremo derecha que contiene un sólo dígito y el signo. El signo; +, -, o **absoluto**, se localiza en los 4 últimos bits del último octeto de la cifra, con las mismas configuraciones de bits que en FORMATO ZONA (C, D o F).

Ejemplo:

Cifra **159**  
signo

			↓
159+	POSITIVO (+)	15   9C	
159-	NEGATIVO (-)	15   9D	
159	ABSOLUTO (+)	15   9F	

### 6.3 FORMATO BINARIO PURO (SÓLO VALORES NUMÉRICOS)

Los valores numéricos son almacenados en formato binario puro. Es decir, el valor numérico decimal, antes de almacenarse es traducido al sistema numérico en base 2 (ceros y unos).

**Son campos de “longitud fija”** a razón de:

2 Octetos: Media palabra (de 1 a 4 dígitos)

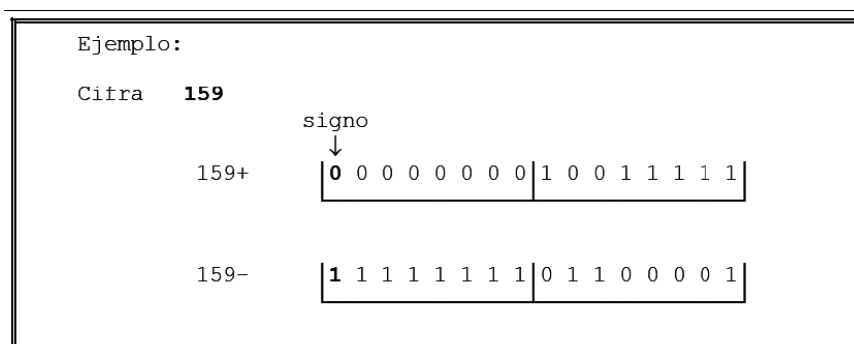
4 Octetos: Una palabra (de 5 a 9 dígitos)

8 Octetos: Doble palabra (de 10 a 18 dígitos)

El signo + o - se localiza en este tipo de campos en el primer bit del primer octeto de la izquierda. Ejemplo:

signo + positivo ==> 0

signo - negativo ==> 1



### 7. CLÁUSULA USAGE

**El formato** de los datos se define en COBOL mediante **la cláusula USAGE**.

La cláusula usage admite tres posibilidades de codificación en función del formato del dato que se desee definir. Estas son DISPLAY, COMPUTACIONAL-3 y COMPUTACIONAL.



### 7.1 USAGE DISPLAY (ALFANUMÉRICOS, ALFABÉTICOS Y NUMÉRICOS)

Indica que el dato que se está definiendo, se utilizará en **FORMATO ZONA**. Puede referirse tanto a campo de clase **A** y **X** como de clase **9**.

En cualquiera de los tres casos se entiende que el dato se aloja en el campo a razón de 1 carácter por octeto.

Ejemplos:

```
02 CODIGO PIC X(5) DISPLAY .
```

Se define el campo CODIGO  
con 5 octetos

--	--	--	--	--

1 2 3 4 5

```
02 CODIGO PIC 9(3) .
```

Se define el campo IMPORTE  
con 3 octetos

--	--	--

1 2 3

**Nota:** Por ausencia de la cláusula USAGE el compilador asume DISPLAY.

### 7.2 USAGE COMPUTATIONAL-3 o COMP-3 (CAMPOS NUMÉRICOS)

Indica que el dato que se está definiendo, se utilizará en **FORMATO EMPAQUETADO**. Sólo se utiliza para campos de clase **9**.

Se entiende que el conjunto de dígitos indicado, se aloja a razón de 2 dígitos por octeto, excepto el primer octeto de la derecha, que aloja sólo uno, reservando el medio octeto derecho para el signo.

Ejemplos:

**01 CAMPO      PIC 9(5)    COMP-3.**

DD	DD	DS
1	2	3

- Se pretende alojar un dato de 5 dígitos en un campo de 3 octetos.

**01 CAMPO-2    PIC S9(9)    COMP-3.**

DD	DD	DD	DD	DS
1	2	3	4	5

- Los 9 dígitos se alojarán en 5 octetos.

### 7.3 USAGE COMPUTATIONAL o COMP. (CAMPOS NUMERICOS)

Indica que el dato que se está definiendo, se utilizará en FORMATO BINARIO PURO. Sólo puede referirse a campos de clase **9**.

Se entiende que el conjunto de dígitos del dato se aloja en formatos predefinidos, teniendo en cuenta el número de dígitos.

Así, si el número de dígitos está comprendido entre 1 y 4, el dato se alojará NECESARIAMENTE en 2 octetos o, lo que es lo mismo, en MEDIA PALABRA.

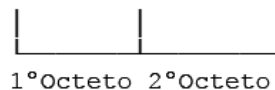
Si el número de dígitos está comprendido entre 5 y 9, el dato se alojará NECESARIA MENTE en 4 octetos o, lo que es lo mismo, en UNA PALABRA.

Si el número de dígitos está comprendido entre 10 y 18, el dato se alojará NECESARIAMENTE en 8 octetos o, lo que es lo mismo, en una DOBLE PALABRA.

Ejemplo 1:

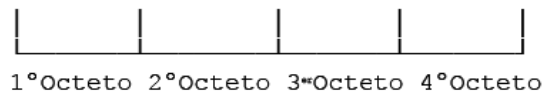
**02 INDICE PIC S9(3) COMP.**

-Se pretende alojar una cifra de no más de tres dígitos y que, por tanto, la estructura de campo que soportará el dato será una MEDIA PALABRA, es decir, 2 OCTETOS.



Ejemplo 2:

**02 VALOR-BINARIO PIC S9(9) COMP.**



**Nota:** El valor del campo se configura en binario puro.

## 8. EL SIGNO EN LOS CAMPOS NUMÉRICOS

El indicativo de que el valor numérico que contenga un campo tiene SIGNO, es una “S” delante del 9 que indica picture numérica.

Ejemplo:

**02 IMPORTE PIC S9(5) .**



Las operaciones aritméticas se realizan algebraicamente. El signo, por tanto, como es lógico, se tiene en cuenta por el ordenador en el momento de realizar cálculos.

Un campo numérico puede ser receptor como resultado de una operación aritmética o como resultado de un movimiento unitario, es decir, si el dato procede de otro campo elemental. En ambas situaciones la presencia o ausencia del indicativo del signo “S” en la descripción de los campos, condiciona que el dato en el campo receptor GANE o PIERDA el SIGNO.

Como norma, es conveniente definir con signo todos aquellos campos numéricos con los que se vayan a realizar operaciones aritméticas o sean utilizados como receptores de otros campos o valores numéricos.

CAMPO-EMISOR	CAMPO-RECEPTOR	RESULTADO
CON SIGNO	CON SIGNO	CON SIGNO
CON SIGNO	SIN SIGNO	PIERDE SIGNO
SIN SIGNO	CON SIGNO	GANAN SIGNO
STN SIGNO	STN SIGNO	STN SIGNO

## 9. LOS DECIMALES EN LOS CAMPOS NUMÉRICOS

Es posible definir el lugar donde interesa colocar EL PUNTO DECIMAL para que las operaciones se realicen de acuerdo con el mismo.

El indicativo del PUNTO DECIMAL es el carácter “ V “. Para dicho carácter no se reserva ningún espacio físico dentro del campo.

El lugar donde estará el punto decimal es “supuesto” por el compilador a partir de la picture declarada del campo.

Ejemplos:

↓

**02 CAMPOA PIC S9(8)V9(4) VALUE +19876823,4.**

contenido    

F1	F2	F3	C4
----	----	----	----

•

- En el lugar marcado con el punto  
(•), se supone que debe estar  
el punto decimal.

↓

**77 CAMPOB PIC S999999V999 COMP-3 VALUE +12345,678.**

contenido    

01	23	45	67	8C
----	----	----	----	----

•

- En el lugar marcado con el punto  
(•), se supone que debe estar  
el punto decimal.

## 10. INICIALIZACIÓN DE VARIABLES. CLAÚSULA VALUE

Los valores se incluyen en los campos de trabajo (dentro de la WORKING-STORAGE SECTION) mediante la cláusula **VALUE**.

La FILE SECTION y la LINKAGE SECTION **no** admiten la cláusula VALUE.

**Esquema de la cláusula:**

**VALUE [ IS ] literal**

**LITERAL:**

En un conjunto de caracteres alfanuméricos o una cifra numérica que se incluye como valor o contenido de un campo mediante la cláusula VALUE en la WORKING-STORAGE SECTION. (El literal asignado a un campo, a través de VALUE ha de estar de acuerdo con el tipo de picture de dicho campo).

Los literales pueden ser NUMÉRICOS y ALFANUMÉRICOS.

### 10.1 LITERAL NUMÉRICO

Es una cifra de **1 a 18 dígitos numéricos**, un punto decimal, si es necesario y el signo operacional **+**, **o**, **-**, si procede. (Los valores numéricos nunca van encerrados entre apóstrofes).

	4	6	7	A	B														
				8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	72
050					02	IMPORTE				PIC	S9 (3)				VALUE	+123.			
060					02	IMPOT1				PIC	S9 (5)				VALUE	+0.			
070					02	V-A				PIC	S9 (3) COMP-3				VALUE	-5 .			
080					02	COD1				PIC	9 (4)				VALUE	3333.			

### 10.2 LITERAL ALFANUMÉRICO

Conjunto de 1 a 160 caracteres alfanuméricos que se incluyen como contenido de un campo lógicamente definido con formato/usage DISPLAY.

#### REGLAS DE CONSTRUCCIÓN:

Un literal alfanumérico se escribe entre apóstrofes (o entrecomillado en función del compilador).

Si el literal excede la línea donde se está codificando, se completará dicha línea hasta la columna 72, inclusive, con parte del texto. En la línea siguiente se codificará una nueva comilla y se completará el texto.

En la columna 7 de la línea nueva se codificará un guión.

Si el literal no se parte, en dos o más líneas, el guión no será necesario.

## Ejemplos. HOJA DE CODIFICACIÓN EN COBOL

	A	B																		
4	6	7	8	12	16	20	24	28	32	36	40	44	48	52	56	60	68	72		
010				02	CAMPO-T				PIC X(6)				VALUE				'AAAAAA'.			
020				02	FILLER				PIC X(20)				VALUE				'ABCABCABCABCABCABCABC			
030	-																'ABC'.			
040				02	NUMERICO-NO				PIC X(10)				VALUE				'0123456789'.			
050				02	MENSA-ERROR				PIC X(45)				VALUE							
060																	'INTENTO DE DAR DE ALTA UN REG. QUE EXISTE'.			

## 10.3 CONSTANTES FIGURATIVAS

Son palabras reservadas a las que el compilador les asigna un VALOR. Este valor se genera como valor inicial de un campo cuando una constante figurativa se asocia al campo mediante la cláusula VALUE.

Las constantes figurativas también se emplean en sentencias de proceso.

Las constantes figurativas son: SPACE/S, HIGH-VALUE/S, LOW-VALUE/S, ZERO, ZEROS o ZEROES y ALL 'caracteres':

**SPACE / SPACES:** Rellena con el valor “**blancos**” el campo al que se asigna.

Ejemplo:									
02	CAMPO1	PIC	X(4)	VALUES	'	'	.		
				40	40	40	40		
Sólo para campos ALFANUMÉRICOS									

**HIGH-VALUE/S:**

Rellena el campo al que se asigna, con el mayor valor del código EBCDIC (FF en hexadecimal) .

Ejemplo:

```
01 SEQACT PIC X(3) VALUE HIGH-VALUES.  
      FF FF FF
```

Sólo para campos ALFANUMÉRICOS.

**LOW-VALUE/S:**

Rellena con el menor valor (00 en hexadecimal) el campo a que se asigna.

Ejemplo:

```
01 SEQACT PIC X(3) VALUE LOW-VALUES.  
      00 00 00
```

Sólo para campos ALFANUMÉRICOS.



**ALL 'carácter/es':**

Rellena con el carácter o caracteres codificado/s el campo al cual se le asigne.

Ejemplos:

```
03 FILLER PIC X(4) VALUE ALL 'B'.
```

B	B	B	B
---	---	---	---

```
03 CAMPO3 PIC X(7) VALUE ALL 'AB'.
```

A	B	A	B	A
C1	C2	C1	C2	C1

AB	AB	AB	AB	AB	AB	AB
----	----	----	----	----	----	----

Sólo para campos ALFANUMÉRICOS.

**ZERO/ZEROS/ZEROES:** Rellena con cero o ceros el campo al cual se le asigna como valor.

Ejemplos:

```
77 CODIGO PIC X(3) VALUE ZEROS.
```

F0	F0	F0
----	----	----

```
77 CAMPO PIC X(3) VALUE ZERO.
```

F0	40	40
----	----	----

```
02 IMPORTE PIC S9(3) VALUE ZERO.
```

F0	F0	F0
----	----	----

```
02 CANTIDAD PIC S9(3) COMP-3 VALUE ZEROES.
```

00	0F
----	----

NOTA: En estos dos últimos ejemplos se obtiene el mismo resultado con ZERO, ZEROS o ZEROES.

Para campos ALFANUMERICOS y NUMERICOS.

## 11. EDICIÓN DE CAMPOS NUMERICOS (MASCARAS DE EDICION)

Los campos numéricos, cuando se listan, puede ser que necesiten ser editados.

**Editar significa:**

- Suprimir ceros no significativos.
- Incluir comas de separación cada tres cifras entre ras.
- Separar con punto los enteros de los decimales, si los hay.

Para editar un campo el programador deberá describir, en el campo de salida **del listado**, el modelo que servirá de base para realizar dicha edición.

Si la cláusula DECIMAL-POINT IS COMMA está codificada, el punto decimal debe ser sustituido por una coma y las comas de los miles por puntos.

### REGLAS PARA ESCRIBIR LOS MODELOS DE EDICIÓN MÁS UTILIZADOS

- El carácter “9” define una posición donde se colocará un dígito numérico.
- Para suprimir ceros no significativos y reemplazar los por blancos, se escribirá una “Z” en cada posición de dígito a suprimir.
- La sustitución de ceros por blancos finaliza al terminar las “Z” o al llegar al primer dígito significativo.
- Las “Z” deben ir, siempre, delante de los “9”. Una vez escrito un “9” no podrá escribirse ninguna “Z”.
- Para insertar comas y el punto, hay que ponerlos, en el modelo, donde procedan.

#### **El punto o la**

**coma como máscara si ocupan espacio en memoria (1 octeto).**

\_ El punto y la coma, cuando van entre “Z”, se editarán, si van precedidos de una cifra significativa.

- Para editar el **signo**, hay que escribir un “-” detrás del último carácter del modelo. Si el signo es positivo, se editará un espacio en blanco en su lugar.

**Ejemplos:**

Sea el campo:

F0	F0	F0	F3	F4	F5	F6	F7	D8
----	----	----	----	----	----	----	----	----

 Con **PICTURE PIC S9(7)V99**.

Se quiere editar un campo para que el valor en el listado o en pantalla se visualice de la siguiente forma: 3,456.78 y signo (signo, podrá ser + ó - )

**SE PUEDEN CONSTRUIR PARA ESTO VARIOS MODELOS:****1. El modelo podría ser: Z,ZZZ,ZZZ.ZZ-**

- Las “Z” harán que no se impriman los ceros no significativos.
- La coma de los millones no se editará, porque el número no tiene millones.
- El signo no se imprimirá, porque el número es positivo.

El modelo anterior tiene el inconveniente de que, si el número a editar fuera todo ceros, el resultado en salida sería blancos. Esto podrá plantear dudas sobre el contenido del campo a editar ya que si no aparece nada en el campo editado, podría ser por que el campo emisor estaba “vacío” ó con ceros.

**2. Otro modelo, podría ser: Z,ZZZ,ZZ9.99**

Este modelo, suele preferirse al anterior, ya que obliga a editar, como mínimo, la cifra de las unidades y las cifras decimales. Así pues, en el caso de que todo el campo emisor sean ceros, se editaría **0.00**.

**3. Si el modelo elegido fuese: 9,999,999.99-**

Los “9” obligarían al sistema operativo a editar, necesariamente, la cifra que corresponda a la posición que ocupan, aunque sean ceros.

Y en tal caso, el resultado de la edición, sería: 0,003,456.78-

Este formato, no sería válido, para conseguir lo que deseaba en el ejemplo, ya que se ven los ceros no significativos.

Hay varios caracteres más susceptibles de ser utilizados en máscaras de edición (**\$, \*, +, B, 0, CR, DB, /, P, V**) que se auto explican en el cuadro de la página siguiente.

**NOTA:** Aunque transparente al programador es interesante saber que, el compilador empleará el modelo de edición seleccionado para reservar dos áreas de trabajo:

1 - Una para almacenar el modelo.

2 - Otra para efectuar la edición.

Al realizarse, en el programa, una instrucción MOVE del campo numérico al campo de salida, se ejecutan tres movimientos:

1º) Se transfiere el modelo al campo en el que se realizará la edición.

2º) Se transfieren los datos del campo de origen al campo de edición y se editan.

3º) El resultado, ya editado, se transfiere al campo del listado.

# PROGRAMACIÓN CON LENGUAJE COBOL

## REPRESENTACIÓN DE CARACTERES Y PALABRAS COBOL RESERVADAS

## Representación de Caracteres EBCDIC

Extended Binary Coded Decimal Interchange Code. (Código de Intercambio de información Decimal Codificado en Binario Extendido).

	Los caracteres	El compilador lo traduce...	Su valor decimal	Su valor hexadecimal
Caracteres especiales	LOW-VALUE	0000 0000	00	00
	SPACE	0100 0000	64	40
	.	0100 1011	75	4B
	<	0100 1100	76	4C
	(	0100 1101	77	4D
	+	0100 1110	78	4E
	\$	0101 1011	91	5B
	*	0101 1100	92	5C
	)	0101 1101	93	5D
	-	0110 0000	96	60
	/	0110 0001	97	61
	,	0110 1011	07	6B
	>	0110 1110	10	6E
	'	0111 1101	25	7D
	=	0111 1110	26	7E
Caracteres alfabéticos	(a-z)	(minúsculas)	(129-169)	(81-A9)
	A	1100 0001	93	C1
	B	1100 0010	94	C2
	C	1100 0011	95	C3
	D	1100 0100	96	C4
	E	1100 0101	97	C5
	F	1100 0110	98	C6
	G	1100 0111	99	C7
	H	1100 1000	00	C8
	I	1100 1001	01	C9
	J	1101 0001	09	D1
	K	1101 0010	10	D2
	L	1101 0011	11	D3
	M	1101 0100	12	D4
	N	1101 0101	13	D5
	O	1101 0110	14	D6
	P	1101 0111	15	D7
	Q	1101 1000	16	D8
	R	1101 1001	17	D9
	S	1110 0010	26	E2
	T	1110 0011	27	E3
	U	1110 0100	28	E4
	V	1110 0110	29	E5
	W	1110 0110	30	E6
	X	1110 0111	31	E7
	Y	1110 1000	32	E8
	Z	1110 1001	33	E9

	Los caracteres	El compilador lo traduce...	Su valor decimal	Su valor hexadecimal
<b>Caracteres numéricos</b>	0	1111 0000	40	F0
	1	1111 0001	41	F1
	2	1111 0010	42	F2
	3	1111 0011	43	F3
	4	1111 0100	44	F4
	5	1111 0101	45	F5
	6	1111 0110	46	F6
	7	1111 0111	47	F7
	8	1111 1000	48	F8
	9	1111 1001	49	F9
	HIGH-VALUE	1111 1111	55	FF

## ASCII

**ASCII** : American Standard Code for Information Interchange.

(Código Normalizado Americano para Intercambio de Información).

	Los caracteres	El compilador lo traduce...	Su valor decimal	Su valor hexadecimal
<b>Caracteres especiales</b>	LOW-VALUE	0000 0000	00	00
	SPACE	0100 0000	64	40
	A	0010 0000	32	20
	\$	0010 0100	36	24
	'	0010 0111	39	27
	(	0010 1000	40	28
	)	0010 1001	41	29
	*	0010 1010	42	2A
	+	0010 1011	43	2B
	,	0010 1100	44	2C
	-	0010 1101	45	2D
	.	0010 1110	46	2E
	/	0010 1111	47	2F
<b>Caracteres numéricos</b>	0	0011 0000	48	30
	1	0011 0001	49	31
	2	0011 0010	50	32
	3	0011 0011	51	33
	4	0011 0100	52	34
	5	0011 0101	53	35
	6	0011 0110	54	36
	7	0011 0111	55	37
	8	0011 1000	56	38
	9	0011 1001	57	39
<b>Caracteres especiales</b>	<	0011 1100	60	3C
	=	0011 1101	61	3D
	>	0011 1110	62	3E

	Los caracteres	El compilador lo traduce...	Su valor decimal	Su valor hexadecimal
<b>Caracteres alfabéticos</b>	A	0100 0001	65	41
	B	0100 0010	66	42
	C	0100 0011	67	43
	D	0100 0100	68	44
	E	0100 0101	69	45
	F	0100 0110	70	46
	G	0100 0111	71	47
	H	0100 1000	72	48
	I	0100 1001	73	49
	J	0100 1010	74	4A
	K	0100 1011	75	4B
	L	0100 1100	76	4C
	M	0100 1101	77	4D
	N	0100 1110	78	4E
	O	0100 1111	79	4F
	P	0101 0000	80	50
	Q	0101 0001	81	51
	R	0101 0010	82	52
	S	0101 0011	83	53
	T	0101 0100	84	54
	U	0101 0101	85	55
	V	0101 0110	86	56
	W	0101 0111	87	57
	X	0101 1000	88	58
	Y	0101 1001	89	59
	Z	0101 1010	90	5A
	(a-z)	(minúsculas)	(97-122)	61-7A)
	HIGH-VALUE	1111 1111	55	FF



### Palabras COBOL reservadas

Se trata de palabras con **sentido fijo** en el lenguaje COBOL. Por tanto, serán utilizables solamente dentro de un entorno determinado. A continuación se detallan las palabras reservadas en COBOL II.

-	C 01	CONTROL
(	C 02	CONTROLS
)	C03	CONVERTING
*	C04	COPY
**	C05	CORE-INDEX
/	C06	CORR
+	C07	CORRESPONDING
<	C09	COUNT
<=	C10	CSP
=	C11	CURRENCY
>	C12	CURRENT-DATE
>=	CALL	DATA
ACCEPT	CANCEL	DATE
ACCESS	CBL	DATE-COMPILED
ACTUAL	CD	DATE-WRITTEN
ADD	CF	DAY
ADVANCING	CH	DE
AFTER	CHANGED	DEBUG
ALL	CHARACTER	DEBUG-CONTENTS
ALPHABETIC	CHARACTERS	DEBUGGING
ALSO	CLOSE	DEBUG-ITEM
ALTER	CODE	DEBUG-LINE
ALTERNATE	CODE-SET	DEBUG-NAME
AND	COLLATING	DEBUG-SUB-1
APPLY	COLUMN	DEBUG-SUB-2
ARE	COMMA	DEBUG-SUB-3
AREA	COMMUNICATION	DECIMAL-POINT
AREAS	COMP	DECLARATIVES
ASCENDING	COMP-1	DELETE
ASSIGN	COMP-2	DELIMITED
AT	COMP-3	DELIMITER
AUTHOR	COMP-4	DEPENDING
BEFORE	COMPUTATIONAL	DESCENDING
BEGINNING	COMPUTATIONAL-1	DESTINATION
BLANK	COMPU TATIONAL-2	DETAIL
BLOCK	COMPUTATIONAL-3	DIS
BOTTOM	COMPUTATIONAL-4	DISABLE
BY	COMPUTE	DISPLAY
	CONFIGURATION	DISPLAY-ST
	CONSOLE	DIVIDE
	CONTAINS	DIVISION

DOWN	FILLER	LINAGE-COUNTER
DUPLICATES	FINAL	LINE
DYNAMIC	FIRST	LINE-COUNTER
EGI	FOOTING	LINES
EJECT	FOR	LINKAGE
ELSE	FROM	LOCK
EMI	GENERATE	LOW-VALUE
ENABLE	GIVING	LOW-VALUES
END	GO	MEMORY
END-ADD	GREATER	MERGE
END-CALL	GROUP	MESSAGE
END-DELETE	HEADING	MODE
END-DIVIDE	HIGH-VALUE	MODULES
END-EVALUATE	HIGH-VALUES	MORE-LABELS
END-IF	ID	MOVE
ENDING	IDENTIFICATION	MULTIPLE
END-MULTIPLY	IF	MULTIPLY
END-OF-PAGE	IN	NAMED
END-PERFORM	INDEX	NATIVE
END-READ	INDEXED	NEGATIVE
END-RECIEVE	INDICATE	NEXT
END-RETURN	INITIAL	NO
END-REWRITE	INITIALIZE	NOMINAL
END-SEARCH	INITIATE	NOT
END-START	INPUT	NOT AT END
END-STRING	INPUT-OUTPUT	NOT INVALID KEY
END-UNSTRING	INSERT	NOT ON SIZE ERROR
END-WRITE	INSPECT	NOTE
ENTER	INSTALLATION	NUMBER
ENTRY	INTO	NUMERIC
ENVIRONMENT	INVALID	OBJECT-COMPUTER
EOP	I-O	OCCURS
EQUAL	I-O-CONTROL	OF
ERROR	IS	OFF
ESI	JUST	OMITTED
EVALUATE	JUSTIFIED	ON
EVERY	KEY	OPEN
EXAMINE	LABEL	OPTIONAL
EXCEPTION	LAST	OR
EXHIBIT	LEADING	ORGANIZATION
EXIT	LEAVE	OTHERWISE
EXTEND	LEFT	OUTPUT
FD	LENGTH	OVERFLOW
FILE	LESS	PAGE
FILE-CONTROL	LIMIT	PAGE-COUNTER
FILE-LIMIT	LIMITS	PASSWORD
FILE-LIMITS	LINAGE	PERFORM

PF	REVERSED	STOP
PH	REWIND	STORE
PIC	REWRITE	STRING
PICTURE	RF	SUB-QUEUE-1
PLUS	RH	SUB-QUEUE-2
POINTER	RIGHT	SUB-QUEUE-3
POSITION	ROUNDED	SUBTRACT
POSITIONING	RUN	SUM
POSITIVE	S01	SUPPRESS
PROCEDURE	S02	SYMBOLIC
PROCEDURES	SAME	SYNC
PROCEED	SD	SYNCHRONIZED
PROCESSING	SEARCH	SYSIN
PROGRAM	SECTION	SYSOUT
PROGRAM-ID	SECURITY	SYSPUNCH
QUEUE	SEEK	TABLE
QUOTE	SEGMENT	TALLY
QUOTES	SEGMENT-LIMIT	TALLYING
RANDOM	SELECT	TAPE
RD	SELECTIVE	TERMINAL
READ	SEND	TERMINATE
READY	SENTENCE	TEXT
RECEIVE	SEPARATE	THAN
RECORD	SEQUENCE	THEN
RECORD-OVERFLOW	SEQUENTIAL	THROUGH
RECORDS	SET	THRU
REDEFINES ç	SIGN	TIME
REEL	SIZE	TIME-OF-DAY
REFERENCES	SKIP-1	TIMES
RELATIVE	SKIP-2	TO
RELEASE	SKIP-3	TOP
RELOAD	SORT	TOTALED
REMAINDER	SORT-CORE-SIZE	TOTALING
REMARKS	SORT-FILE-SIZE	TRACE
REMOVAL	SORT-MERGE	TRACK-AREA
RENAMES	SORT-MESSAGE	TRACK-LIMIT
REORG-CRITERIA	SORT-MODE-SIZE	TRACKS
REPLACING	SORT-RETURN	TRAILING
REPORT	SOURCE	TYPE
REPORTING	SOUR-COMPUTER	UNIT
REPORTS	SPACE	UNSTRING
REREAD	SPACES	UNTIL
RERUN	SPECIAL-NAMES	UP
RESERVE	STANDARD	UPON
RESET	STANDARD-1	UPSI-0
RETURN	START	UPSI-1
RETURN-CODE	STATUS	UPSI-2

UPSI-3 UPSI-4 UPSI-5 UPSI-6 UPSI-7 USAGE USE USING VALUE VALUES VARYING WHEN WHEN-COMPILED WITH WITH TEST AFTER WORDS WORKING-STORAGE WRITE WRITE-ONLY ZERO ZEROES ZEROS		
---	--	--

**Hoja de Codificación en COBOL**

Pág.	Programa				Sistema				Hoja Nº de															
13	Programador				Fecha				Identificación 73 80															
LIN	A				B																			
4	5	6	7	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72				
001																								
002																								
003																								
004																								
005																								
006																								
007																								
008																								
009																								
010																								
011																								
012																								
013																								
014																								
015																								
016																								
017																								
018																								
019																								
020																								
021																								
022																								
023																								
024																								
025																								
026																								
027																								
028																								
029																								
030																								
031																								
032																								
033																								

# DESCRIPCIÓN DE FICHEROS

## 1. DATA DIVISION

Dentro de la DATA DIVISION, **los datos**, se agrupan por SECCIONES, en función del origen o utilidad de esos datos.

Para cada programa se definirán únicamente aquellas secciones que sean necesarias.

### 1.1 SECCIONES DE LA DATA DIVISION

#### - FILE SECTION

Para definición de las características de los datos procedentes o con destino a FICHEROS.

#### - WORKING-STORAGE SECTION

Para la definición de todos los datos intermedios o de trabajo que se consideren necesarios para un adecuado procesamiento de los datos.

#### - LINKAGE SECTION

Para definir datos comunes a otro programa con el que se enlaza.

## 2. FILE SECTION

### FUNCIÓN

Descripción de los FICHEROS DE ENTRADA y de SALIDA que inter vienen en el programa.

**PECULIARIDADES DE LA FILE SECTION:**

- \* Se define un NIVEL **FD** POR CADA FICHERO.
- \* En FILE SECTION no se utiliza la CLÁUSULA VALUE.
- \* Mediante la cláusula DATA se define **1** ó **n** tipos de registro.
- \* Cada tipo de REGISTRO de UN FICHERO se define a nivel 01.
- \* Se puede utilizar la cláusula REDEFINES a partir de nivel 02.
- \* Es la primera sección de la DATA DIVISION.

**3. NIVEL FD****FD nombre-fichero**

Donde “nombre-fichero” es el **nombre-interno** del fichero del cual se va a realizar posteriormente la definición del registro. Este fichero tuvo que ser declarado con anterioridad en la INPUT-OUT PUT SECTION de la ENVIRONMENT DIVISION, en la posición del nombre-interno.

Con este nivel, se le indica al compilador que “comienza” la definición del registro y de las características del registro de ese fichero.



#### 4. MODALIDAD DE GRABACIÓN

Esta cláusula indica el modo en que los registros del FICHERO a definir, están grabados o se van a grabar en el dispositivo de almacenamiento.

(No es necesaria y en la próxima revisión de COBOL desaparecerá).

Esquema de la cláusula:

**RECORDING MODE IS modo**

**modo:**

Los modos pueden ser:

**F** Indica que la longitud de todos los registros es la misma.

**V** Indica que la longitud de los registros es VARIABLE.

En las 4 primeras posiciones de cada registro se indica su longitud.

**U** Indica que la longitud de los registros no es fija y además no se especifica.

**NOTA:** Por ausencia de esta cláusula, asume **V**.

#### 5. LONGITUD DEL REGISTRO LÓGICO

Esta cláusula indica el número de caracteres (número de octetos) del registro lógico.

No es necesaria y en la próxima revisión de COBOL desaparecerá.

Esquema de la cláusula:

**RECORD CONTAINS n1 [TO n2] CHARACTERS**

**n1:** Es el número de posiciones del registro lógico.

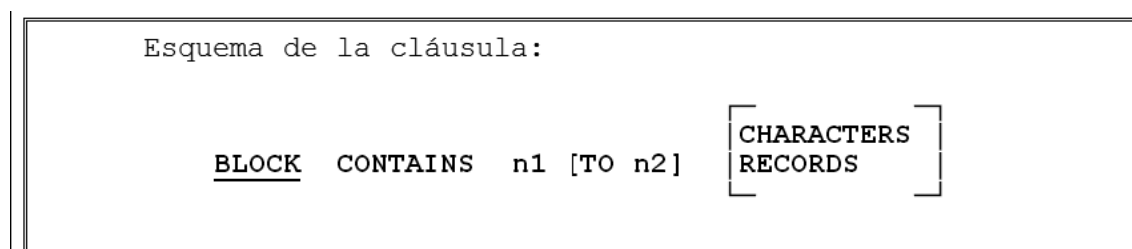
**n2:** Si los registros tienen distintas longitudes, entonces **n1** sería la longitud del registro MENOR y **n2** la longitud del registro MAYOR.

## 6. LONGITUD DEL BLOQUE FÍSICO

Esta cláusula indica o bien el número de registros por bloque, o bien el número de caracteres por bloque en el fichero.

Es la unidad de transferencia de información, entre el dispositivo de almacenamiento del fichero y la memoria principal, durante la ejecución del programa. No es necesaria y en la próxima revisión de COBOL desaparecerá.

Esquema de la cláusula:



**n1:** Es el número de registros lógicos por bloque físico.

**n2:** Si los registros tienen modo **V**, entonces **n1** sería la longitud del bloque menor y **n2** la longitud del mayor.

Si no se codifica esta cláusula, para ficheros SECUENCIALES, asume por defecto, la longitud más adecuada, en función del tipo de soporte físico real usado para el almacenamiento del fichero.

Para los ficheros INDEXADOS y RELATIVOS el BLOCK **NO** se codifica.

### NOTAS:

Si se desea, para ficheros SECUENCIALES, con el fin de independizar el programa de los datos se puede poner:

### BLOCK CONTAINS Ø RECORDS.

En caso de codificar el bloque físico para un fichero que vaya a procesarse en el programa, el Sistema Operativo lo compara con el bloque codificado en el JCL. Si el fichero ya existe, este dato se compara también con el del JCL o, en su defecto, con el bloque existente en la etiqueta del fichero.

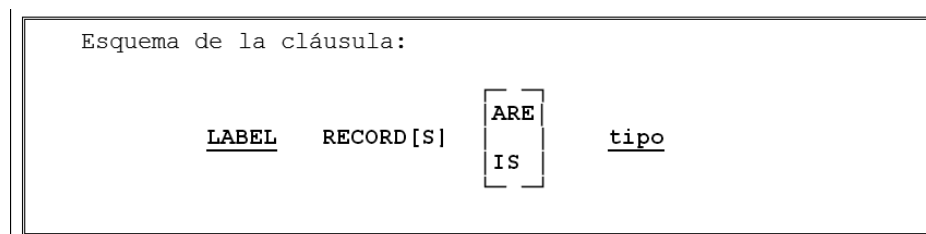
En caso de no coincidir ambos, el Sistema Operativo avisará dando un error en fase de ejecución, al intentar abrir el fichero.

## 7. ETIQUETAS DE FICHEROS

Los FICHEROS grabados en DISCO o Cinta Magnética disponen de registros especiales de identificación propia, denominados ETIQUETAS.

Estos registros son grabados al principio y al final de fichero por el Sistema Operativo.

Esquema de la cláusula:



**TIPO:** Hay dos tipos de etiquetas:

**STANDARD:** Etiquetas generadas automáticamente por el Sistema.

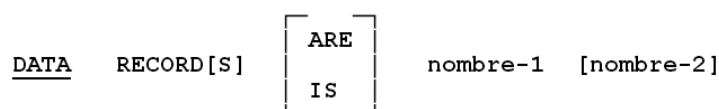
**OMITTED:** Ausencia de etiquetas.

\* La codificación LABEL a partir de la revisión del año 85, es OPCIONAL, y si no se codifica, asume STANDARD para todos los ficheros.

## 8. NOMBRE DEL REGISTRO

Mediante esta cláusula se puede dar nombre al registro o a los registros del FICHERO.

Esquema de la cláusula:



**nombre-1:** En el supuesto de que exista un sólo tipo de registro.

**nombre-2:** Si existen más tipos de registros en el Fichero.

## 9. EJEMPLO DE CODIFICACIÓN DE LA FILE SECTION

	A	B	
4 6 7	8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72	
010	DATA	DIVISION.	
020	*	*****	
025			
030	FILE	SECTION.	
040	*	=====	
045			
050	FD	FICHERO-VENTAS	
060		RECORDING MODE	IS F
060		RECORD CONTAINS	80 CHARACTERS
070	*	BLOCK CONTAINS	0 RECORDS
080		LABEL RECORD	IS STANDARD
090		DATA RECORD	IS REGISTRO-DE-VENTAS.
100			
110	01	R-E.	
120		02 NUMERO-DE-VENTA-E	PIC X(5).
130		02 ZONA-VENTA-E.	
140		03 PROVINCIA-E	PIC X(2).
150		03 AGENCIA-E	PIC X(4).
160		03 VENDEDOR	PIC X(3).
180		02 CLIENTE.	
190		03 CODIGO-CLIENTE	PIC 9(4).
200		03 RAZON-SOCIAL.	
210		04 DIRECCION	PIC X(30).
220		04 TFNO.	
230		05 NUM-TFNO	PIC 9(7).
240		05 EXT-TFNO	PIC 9(3).
260		02 ARTICULO.	
270		03 CODIGO-ARTICULO	PIC X(6).
280		03 PRECIO-UNITARIO	PIC S9(8)V9(2) COMP-3.
290		03 CANTIDAD	PIC S9(5) COMP-3.
300		03 FECHA.	
310		04 AÑO	PIC X(4).
320		04 MES	PIC X(2).
330		04 DIA	PIC X(2).
350			
360			
=====			
050	FD	FICHERO-VENTAS	
060		RECORDING MODE	IS F
060		RECORD CONTAINS	80 CHARACTERS
070	*	BLOCK CONTAINS	0 RECORDS
080		LABEL RECORD	IS STANDARD
090		DATA RECORD	IS REGISTRO-DE-VENTAS.
100			
110	01	REGISTRO-DE-VENTAS	PIC X(80).
=====			
050	FD	FICHERO-VENTAS.	
110	01	REGISTRO-DE-VENTAS	PIC X(80).
=====			
110	01	R-E.	
120		02 NUMERO-DE-VENTA-E	PIC X(5).
130		02 ZONA-VENTA-E.	
140		03 PROVINCIA-E	PIC X(2).
150		03 AGENCIA-E	PIC X(4).

```

160 |      03 VENDEDOR          PIC X(3) .
180 |      02 CLIENTE.
190 |      03 CODIGO-CLIENTE   PIC 9(4) .
200 |      03 RAZON-SOCIAL.
210 |          04 DIRECCION     PIC X(30) .
220 |          04 TFNO.
230 |              05 NUM-TFNO  PIC 9(7) .
240 |              05 EXT-TFNO  PIC 9(3) .
260 |      02 ARTICULO.
270 |      03 CODIGO-ARTICULO   PIC X(6) .
280 |      03 PRECIO-UNITARIO   PIC S9(8)V9(2) COMP-3.
290 |      03 CANTIDAD          PIC S9(5) COMP-3.
300 |      03 FECHA.
310 |          04 AÑO           PIC X(4) .
320 |          04 MES           PIC X(2) .
330 |          04 DIA           PIC X(2) .
350 |
360 |
370 | FD LISTADO-DE-SALIDA
060 | RECORDING MODE          IS F
380 | RECORD CONTAINS        132 CHARACTERS
390 | LABEL RECORD           IS OMITTED
400 | DATA RECORD           IS REG-LISTADO.
420 | 01 REG-LISTADO          PIC X(132) .
410 |
410 |
420 | 01 REG-LISTADO.
430 |     02 CAMPO-SALTO      PIC X.
440 |     02 FILLER           PIC X(132) .
450 |
460 |
470 | * Las cláusulas de la FD no son imprescindibles pero, por claridad,
480 | * si se desea, se pueden exponer las características del fichero:
490 | *   - Todos los registros tienen la misma longitud.
500 | *   - La longitud del registro es de 230 Bytes.
510 | *   - El Bloque Físico lo calculará el S.O.
520 | *   - Quedará guardado en Disco.

530 | FD FICHERO-MAESTRO-NUEVO.
540 |
550 | 01 REGISTRO-MAESTRO-NUEVO PIC X(230) .
560 |

```

## 10. SENTENCIA COPY E INCLUSIÓN DE MÓDULOS FUENTE

La sentencia COPY ordena al compilador que recupere un fichero de una librería de **módulos fuentes** y lo inserte en el lugar donde se codificó la sentencia.

### FORMATO DE LA SENTENCIA

**COPY nombre-de-miembro**

**[ IN nombre-de-librería ]**

Normalmente, las librerías disponibles para el compilador se definen a través de sentencias de orden/control de tarea al ejecutar el compilador y la cláusula **IN**, normalmente no es necesaria, porque el compilador busca automáticamente la librería especificada por defecto para el compilador.

### VENTAJAS QUE APORTA LA SENTENCIA COPY

« **Ahorro de tiempo** de codificación a los programadores, ya que las sentencias sólo es necesario escribirlas una vez aunque se utilicen en muchos programas.

« **Se evitan posibles errores** e interpretaciones equívocas respecto a PICTURE, USAGE y formato de los registros del archivo.

« El desarrollo y mantenimiento de los programas se hace más fácil y rápido ya que los nombres de los archivos y de los campos se **normalizan** de un programa a otro.

« El uso de la sentencia COPY hace más fácil las modificaciones de aplicaciones, porque sólo se necesita cambiar **una sola vez** una FD, una descripción de registro, una rutina, etc. Todos los programas a los que afecte el cambio, sólo deben ser recompilados sin ninguna modificación directa en el programa fuente.

# CREACIÓN Y TRATAMIENTO DE TABLAS

## 1. CLÁUSULA REDEFINES

Mediante esta cláusula se puede definir varias veces **la misma zona de memoria**, con **nombre, tipo y formato distintos**.

Es decir, permite definir "UN MISMO CAMPO" con VARIOS NOMBRES DISTINTOS y PICTURES DISTINTAS.

### FACILIDADES:

Además de poder **nombrar la misma zona con distintos nombres** es posible:

- Definir **un mismo campo con distintos formatos (USAGE)**.
- Definir **un mismo campo cambiando el tipo** del mismo.
- Definir **un área parcelándola en distintas longitudes**.

### ESQUEMA DE LA CLÁUSULA:

**Nivel nom-campo-R REDEFINES nom-campo [ PIC... ].**

**Nivel:** Se puede codificar bajo cualquier nivel, excepto bajo los niveles 66 y 88.

**nom-campo-R:** Será el **nuevo nombre** con que se puede localizar una zona de memoria que ya se definió anteriormente con otro nombre.

**nom-campo:** Es el nombre que se asignó antes al campo que se está redefiniendo.



**UTILIZACIÓN:**

La cláusula REDEFINES se utiliza en WORKING-STORAGE SECTION a **cualquier nivel**, y en la FILE SECTION a cualquier nivel que **no sea 01**.

El redefinidor se sitúa debajo del redefinido, inmediatamente o después de los niveles dependientes del redefinido. Tanto el redefinidor como el redefinido deben tener **el mismo nivel**.

No está permitido utilizar la cláusula VALUE en los nombres de campo redefinidos a continuación y en dependencia de la cláusula REDEFINES.

Las longitudes de las zonas redefinidas es conveniente que sean las mismas.

## EJEMPLOS DE REDEFINICIONES:

	A	B
4 6 7 8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72	
005		
010		02 CANT PIC S9(5) COMP-3.
020		02 CANTR REDEFINES CANT PIC X(3).
010		02 CANT PIC 9(5).
020		02 CANTR REDEFINES CANT PIC X(5).
030	* ***	El contenido de una misma zona de memoria es considerado como
031	*	numérico o como alfanumérico.
032	*	iiiiii Ojo, con los cambios del formato (USAGE) !!!!!
033		
040	01	GRUPO.
050		02 E PIC X(10).
060		02 F PIC X(10).
070		
080	01	GRUPOR REDEFINES GRUPO.
090		02 ER PIC X(14).
100		02 FR PIC X(6).
101	* ****	Una misma zona se subdivide con distintas longitudes.
102		
110		
120	01	LITERALES-DE-DIAS.
		02 TROZO PIC X(63) VALUE 'LUNES MARTES MIERCOLES...'
	01	LITERALES-DE-DIAS.
130		02 FILLER PIC X(9) VALUE 'LUNES'.
140		02 FILLER PIC X(9) VALUE 'MARTES'.
150		02 FILLER PIC X(9) VALUE 'MIERCOLES'.
160		02 FILLER PIC X(9) VALUE 'JUEVES'.
170		02 FILLER PIC X(18) VALUE 'VIERNES SABADO'.
180		02 FILLER PIC X(9) VALUE 'DOMINGO'.
190		
200	01	TABLA-DIAS REDEFINES LITERALES-DE-DIAS.
210		02 DIA PIC X(9) OCCURS 7.
220		
230	* ****	Una zona de memoria es subdividido en trozos iguales.
240		

## 2. CLÁUSULA OCCURS

Mediante esta cláusula **se asigna un nombre común a varios campos contiguos**. Para que esto sea posible, estos campos además de contiguos deberán tener la **misma** clase, longitud y usage.

La cláusula OCCURS permite definir VARIOS CAMPOS (varias zonas de memoria) CON EL MISMO NOMBRE; en cambio, la cláusula REDEFINES permite asignar varios nombres a una misma zona de memoria.

La cláusula OCCURS no se permite a nivel 01. Y en un mismo nivel la cláusula OCCURS y VALUE a la vez son incompatibles.

### ESQUEMA DE LA CLÁUSULA:

**02 nombre-elemento [PIC ...] OCCURS n-entero [TIMES].**

#### **nombre-elemento:**

Es el nombre común, también denominado Elemento, asignado a una zona de memoria, la cual se repite a continuación un número de veces.

#### **n-entero TIMES:**

Es el número de campos contiguos que se deben crear.

No está permitido utilizar la cláusula VALUE en los nombres de campo definidos a continuación y en dependencia de la cláusula OCCURS, en las “releases” de COBOL anteriores al año 85.

#### **PIC ... :**

Se puede codificar la PICTURE de n-campo si éste no es un campo compuesto.

### 3. DEFINICIÓN DE TABLAS

Al conjunto de “**campos contiguos**” que tienen el **mismo nombre**, la **misma clase**, la **misma longitud** y el **mismo usage** se les denomina TABLAS (o matrices).

A cada uno de estos campos contiguos iguales, se les denomina ELEMENTOS.

Luego, una tabla, estará formada por un conjunto de ELEMENTOS.

Para la definición de TABLAS se utilizará la cláusula OCCURS.

Las tablas, en función de cómo este constituido el elemento, se pueden clasificar en SIMPLES y COMPUESTAS; y las tablas compuestas, a su vez, se pueden subdividir en UNIDIMENSIONALES, BIDIMENSIONALES, TRIDIMENSIONALES, etc.

#### 3.1 TABLAS SIMPLES

Una tabla es simple, cuando el campo al que se le asocia la cláusula OCCURS, es un **campo elemental**.

Ejemplo:

**01 TABLA1.**

**02 ELEM OCCURS 4 PIC X(7) .**

Se habrá definido la siguiente zona de memoria:

TABLA1			
ELEM (1)	ELEM (2)	ELEM (3)	ELEM (4)

La longitud de TABLA1 será  $4 \times 7 = 28$  BYTES.

### 3.2 TABLAS COMPUESTAS

Una tabla será compuesta cuando el campo al que se le asocia la cláusula OCCURS, es un **campo compuesto** (o grupo) .

Ejemplo:

**01 TABLA2.**

**02 ELEMEN OCCURS 5.**

**03 COD PIC X(3).**

**03 CAN PIC 9(4).**

TABLA2									
ELEMEN (1)		ELEMEN (2)		ELEMEN (3)		ELEMEN (4)		ELEMEN (5)	
COD (1)	CAN (1)	COD (2)	CAN (2)	COD (3)	CAN (3)	COD (4)	CAN (4)	COD (5)	CAN (5)

La longitud de TABLA2 es  $(3+4) * 5 = 35$  OCTETOS.

### 3.3 TABLAS UNIDIMENSIONALES

Las tablas, en función del número de OCCURS que la integran, pueden ser unidimensionales, bidimensionales o tridimensionales.

Así pues, se denomina UNIDIMENSIONALES a las tablas en la que hay un sólo campo afectado por la cláusula OCCURS.

Con **un solo índice** se puede identificar cualquiera de los campos que la componen.

Ejemplo:

**01 TABLA-UNI.**

**02 ELEMENTO OCCURS 3 TIMES.**

**03 CAMA PIC XX.**

**03 CAMB PIC S99.**

Para la definición de la siguiente zona de memoria:

TABLA-UNI					
ELEMENTO (1)		ELEMENTO (2)		ELEMENTO (3)	
CAMA (1)	CAMB (1)	CAMA (2)	CAMB (2)	CAMA (3)	CAMB (3)

La longitud de TABLA-UNI es  $3 * 4 = 12$  OCTETOS.

### 3.4 TABLAS BIDIMENSIONALES

Se denominan BIDIMENSIONALES, a las tablas en las que un campo afectado por la cláusula OCCURS, es dependiente a su vez de otro campo ya afectado por la misma cláusula.

Es decir, cuando un elemento de la tabla o un campo de dicho elemento es a su vez otra tabla. Para direccionar a cualquiera de los elementos de la tabla más interna, será necesario, utilizar **dos índices**.

Ejemplo:

**01 TABLA-BIDIMENSIONAL.**

**02 MES OCCURS 3.**

**03 SEM OCCURS 4 PIC XX.**

Para la definición de la siguiente zona de memoria:

TABLA-BIDIMENSIONAL											
MES (1)				MES (2)				MES (3)			
SEM1 (1,1)	SEM2 (1,2)	SEM3 (1,3)	SEM4 (1,4)	SEM (2,1)	SEM (2,2)	SEM (2,3)	SEM (2,4)	SEM (3,1)	SEM (3,2)	SEM (3,3)	SEM (3,4)

La longitud de TABLA-BIDIMENSIONAL será  $((2*4) * 3) = 24$  BYTES.

### 3.5 TABLAS TRIDIMENSIONALES

Se consideran TRIDIMENSIONALES, las tablas en las que un campo afectado por la cláusula OCCURS, es dependiente de otro campo afectado por otra cláusula OCCURS, la cual, a su vez, es dependiente de otro campo afectado por otra cláusula OCCURS.

Para identificar/referenciar/direccionar un campo del nivel más interno, serán precisos **tres índices**.

Ejemplo:

**01 TABLA-TRIDIMENSIONAL.**

**02 ARTICULO OCCURS 2.**

**03 CODIGO OCCURS 3.**

**04 PRECIO OCCURS 2 PIC X(4).**

Para la definición de la siguiente zona de memoria:

TABLA-TRIDIMENSIONAL											
ARTICULO (1)						ARTICULO (2)					
CODIGO (1,1)		CODIGO (1,2)		CODIGO (1,3)		CODIGO (2,1)		CODIGO (2,2)		CODIGO (2,3)	
PRECIO (1,1,1)	PRECIO (1,1,2)	PRECIO (1,2,1)	PRECIO (1,2,2)	PRECIO (1,3,1)	PRECIO (1,3,2)	PRECIO (2,1,1)	PRECIO (2,1,2)	PRECIO (2,2,1)	PRECIO (2,2,2)	PRECIO (2,3,1)	PRECIO (2,3,2)

La longitud de esta TABLA-TRIDIMENSIONAL es:

$((4 * 2) * 3) * 2 = 48 \text{ BYTES.}$

### 4. CLÁUSULA RENAMES Y NIVEL 66

Proporciona un nuevo nombre a un campo elemental, un campo compuesto o un grupo de campos elementales. La cláusula RENAMES se codifica asociado al NIVEL 66.

Es similar a la cláusula REDEFINES pero no puede cambiar las pictures de los campos.





## 5. CLÁUSULA BLANK WHEN ZERO

El contenido de un campo, si es ceros, se cambiará por blancos, si al definir un campo se le asigna la cláusula BLANK WHEN ZERO.

Solamente es utilizada en campos elementales **numéricos** o numéricos de **edición**.

No es muy usada, ya que se consigue el mismo resultado, dotando al campo con la PICTURE Z.

### FORMATO:

[ BLANK WHEN ZERO ]

03 CANTIDAD PIC ... BLANK WHEN ZERO.

## 6. CREACIÓN DE TABLAS

Las tablas son conjuntos de campos consecutivos los cuales, tienen el mismo Nombre, además de la misma Clase, Longitud y Usage.

Para la definición de tablas se emplea la cláusula OCCURS. Esta cláusula admite varios formatos más, los cuales, se verán a continuación.

### 6.1. CREACIÓN DE TABLAS VARIABLES

Este formato, permite la creación de tablas, con la posibilidad de que ésta contenga un número variable de elementos para cada una de las distintas ejecuciones del programa.

### FORMATO:

.... OCCURS valor-menor [ TO valor-mayor ]

DEPENDING ON nombre-campo.

**DEPENDING ON:** Esta opción se utiliza cuando **el número** de elementos de la tabla, **no sea conocido de antemano** por el programador sino que depende del valor de otro campo.

El número de elementos, variará en cada ejecución en función del valor de nombre-campo.

**nombre-campo:** Su valor es utilizado para fijar el número de elementos con los que se crea la tabla.

Debe ser numérico entero y debe estar en alguna definición anterior a la de la cláusula OCCURS. **valor-menor y valor-mayor:**

Son dos números enteros que indican cuál será el rango menor y mayor de elementos que la tabla puede contener.

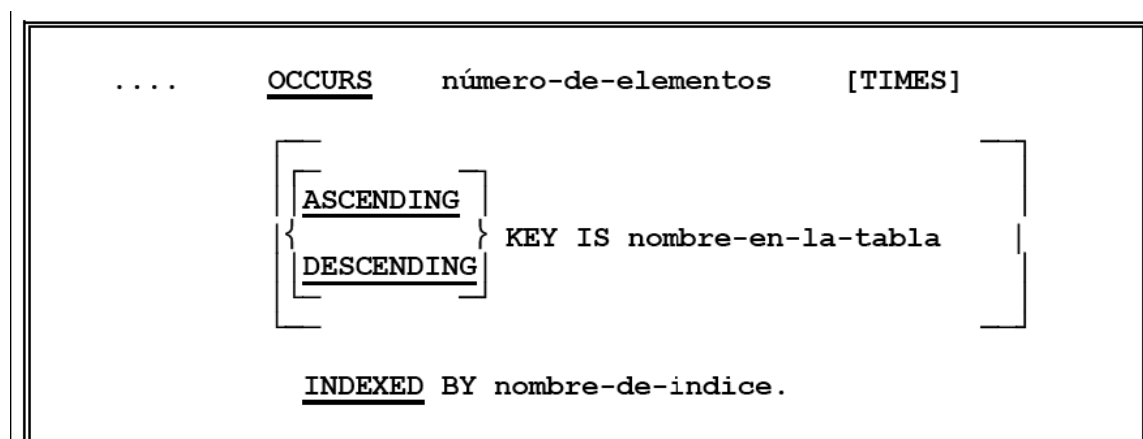
	A	B
4 6 7 8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72	
010	01	REGISTRO-ENTRADA.
020	02	CLAVE-REG PIC X(5) .
030	02	NUM-AÑOS-ENTRADA PIC 9(3) .
040	02	AÑOS OCCURS 50 TO 200
050		DEPENDING ON NUM-AÑOS-ENTRADA.
060	03	CODIGO-AÑO PIC X(4) .
070	03	OTRO-CAMPO PIC X(20) .
080	02	RESTO-REG PIC X(100) .
090		

## 6.2. CREACIÓN DE TABLAS INDEXADAS Y ORDENADAS

Se utiliza para **crear una tabla y un índice asociado** a dicha tabla.

El índice no debe definirse por el programador, ya que lo hace el compilador internamente.

El formato INDEXED BY permite el uso posterior del verbo SEARCH para localizar un elemento dentro de la tabla. (Una **búsqueda secuencial** se efectúa mediante el verbo SEARCH y una **búsqueda dicotómica** mediante el verbo SEARCH ALL).



**ASCENDING/ DESCENDING:** Estas opciones indican que los elementos de la tabla están ordenados en ascendente o descendente en función del contenido de un campo de la tabla (nombre-en-la-tabla).

**INDEXED BY:** Opción que le indica al compilador que **crea un índice** (nombre-de-índice), el cual será utilizado para direccionar los elementos de la tabla.

Ejemplo:

		A	B
4	6	7	8
			12 16 20 24 28 32 36 40 44 48 52 56 60 64
010		01	TABLA-INDEXADA.
020		02	MATERIAL OCCURS 10
030			ASCENDING KEY IS COD-MAT-T
040			INDEXED BY INDICE.
050		03	COD-MAT-T PIC X(3) .
060		03	DESCRI-T PIC X(20) .
070		03	PRECIO-T PIC 9(5) COMP-3.

## 7. VERBO SET

Se emplea para dar valor a un índice, el cuál fue creado por el compilador mediante la cláusula

### INDEXED BY índice.

Este verbo tiene dos formatos, que se muestran a continuación.

#### FORMATO 1:

**SET nombre-de-índice TO valor**

**Nombre-de-índice:** Es el índice a inicializar. (Es el RECEPTOR).

**Valor:** Nombre de campo o literal numérico, con el que se desea inicializar nombre-de-índice.

#### FORMATO 2:

**SET**      nombre-de-índice       $\left\{ \begin{array}{l} \text{UP BY} \\ \text{DOWN BY} \end{array} \right\}$       valor

**UP BY:** Se utilizan para operaciones de **Suma** con índices creados mediante la cláusula INDEXED BY, ya que, el resto de operaciones aritméticas no están permitidas.

**DOWN BY:** Se utilizan para operaciones de **Resta** con índices creados mediante la cláusula INDEXED BY, ya que, el resto de operaciones aritméticas no están permitidas.

Ejemplo:

	4	6	7	8	A	B	12	16	20	24	28	32	36	40	44	48	52	56	60
010							SET		INDICE-1		TO				1.				
020							SET		INDICE-2		UP		BY		5.				
030							SET		INDICE-2		UP		BY		-5.				
040							SET		INDICE-2		DOWN		BY		1.				
050							SET		INDICE-2		DOWN		BY		-1.				
060																			
070																			

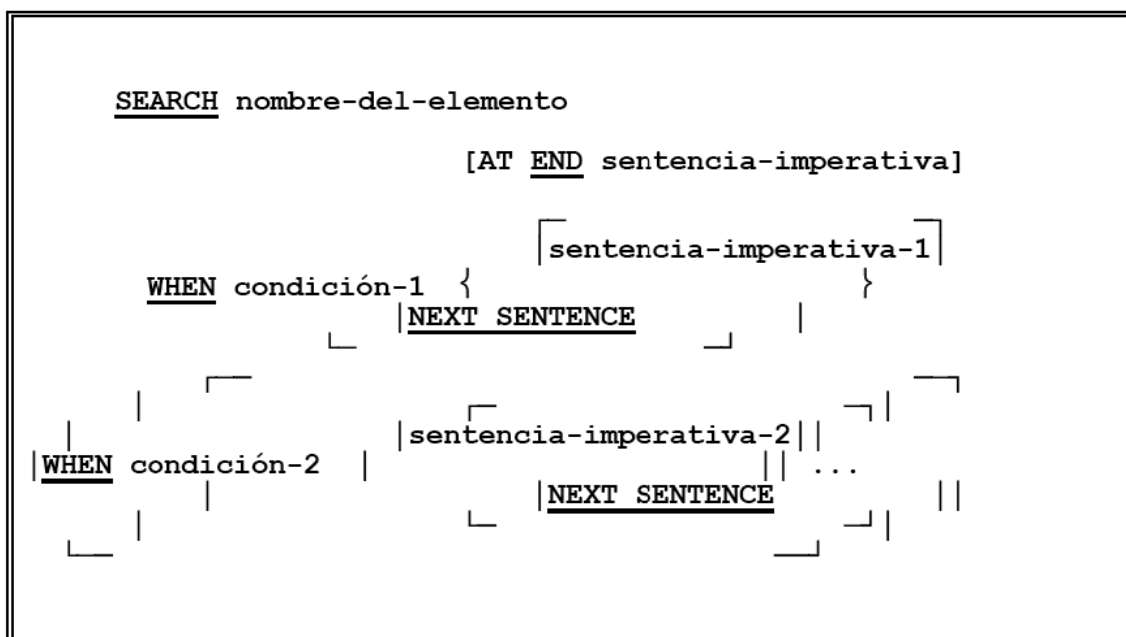
## 8. VERBO SEARCH (BÚSQUEDA SECUENCIAL EN TABLAS)

Realiza una **búsqueda secuencial** en una tabla creada con la opción INDEXED BY. No es necesario que la tabla esté ordenada.

« **Antes de efectuar la búsqueda**, se debe inicializar el índice, asociado a INDEXED BY mediante el verbo SET. El índice deberá ser positivo y mayor que cero. Si el valor fuera superior a número de elementos NO se realiza la búsqueda.

« **La búsqueda finaliza** cuando el contenido de un elemento satisface una condición prefijada o cuando se llega al final de la tabla.

### FORMATO:



**SEARCH:** Verbo que indica búsqueda secuencial.

**nombre-del-elemento:** Es el nombre genérico asignado a todos los elementos de la tabla.

**WHEN condición sentencia-imperativa-1:** Se solicita el análisis de una condición. Se pueden poner varios WHEN y la/s condición/es es/son analizada/s para cada elemento, si no se cumple

ninguna **se incrementa internamente el índice** y se continua hasta que alguna condición se cumple. En tal caso se ejecutan las sentencias asociadas a esa condición y la búsqueda concluye, saltando al siguiente punto.

#### **AT END sentencia-imperativa:**

Instrucciones que se ejecutarán si se lleva al final de la tabla sin que ninguna condición se cumpla.

	A	B
4 6 7 8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72	
010	WORKING-STORAGE SECTION.	
020	*****	
030	01 TABLA-ALMACENES.	
040	02 ALMACEN-T OCCURS 10	INDEXED BY INDICE-1.
050	03 COD-ALMA-T	PIC XX.
060	03 LITE-ALMA-T	PIC X(15) .
070		
080		
090	PROCEDURE DIVISION.	
100	*****	
110	SET INDICE-1	TO 1.
120	SEARCH ALMACEN-T	
130	AT END	PERFORM NO-EXISTE-ALMACEN
140	WHEN COD-ALMA-T(INDICE-1)	= COD-ALMA-E
150	MOVE LITE-ALMA-T(INDICE-1)	TO ALMA-CAB-1.

## 9. VERBO SEARCH ALL (BÚSQUEDA DICOTÓMICA)

Se emplea para realizar **búsqueda binarias o dicotómicas** en tablas creadas con la opción INDEXED BY.

Es **imprescindible** que la tabla esté **ORDENADA** en ascendente o descendente. No es necesario inicializar el índice mediante SET. El sistema gestiona el índice internamente.

### FORMATO:

```
SEARCH ALL nombre-del-elemento  
[AT END sentencia-imperativa]  
  
WHEN condición {  
    sentencia-imperativa  
    NEXT SENTENCE  
}
```

Todo lo dicho en el formato anterior es válido para éste excepto que:

**SEARCH ALL:** Verbo para efectuar búsquedas binarias.

**WHEN condición:** WHEN sólo puede aparecer **una vez** y condición únicamente puede usarse con el operador de relación **EQUAL TO (=)** y con el operador lógico **AND**.

## EJEMPLO:

	A	B
4 6 7	8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72
010	WORKING-STORAGE SECTION.	
020	*=====	
030	01 TABLA-DE-MATERIALES.	
040	02 MATERIAL-T OCCURS 5	
050	ASCENDING KEY IS CODIGO-MATERIAL-T	
060	INDEXED BY INDICE-1.	
070	03 CODIGO-MATERIAL-T PIC XX.	
080	03 CLAVES-DE-PRECIO-T OCCURS 9	
090	ASCENDING KEY IS CLV-PRECIO-T	
100	INDEXED BY INDICE-2.	
110	04 CLV-PRECIO-T PIC X.	
120	04 COEFICIENTE-T PIC 9(3).	
130		
140		
150	PROCEDURE DIVISION.	
160	*=====	
200		
170	SEARCH MATERIAL-T	
180	AT END PERFORM RUTINA-MATERIAL-NO-ENCONTRADO	
180	WHEN CODIGO-MATERIAL-T(INDICE-1) = MATERIAL-MOVIM-E	
190	NEXT SENTENCE.	
200	END-SEARCH.	
180		
210	SEARCH CLAVES-DE-PRECIO-T	
380	AT END PERFORM RUTINA-PRECIO-NO-ENCONTRADO	
320	WHEN CLV-PRECIO-T(INDICE-1 INDICE-2) = CLV-PRE-MOV-E	
400	MOVE COEFICIENTE-T(INDICE-1 INDICE-2) TO COEFIC-LW	
480	END-SEARCH.	
490		
490		
530		

## 10. GESTIÓN DE TABLAS Y VERBO PERFORM VARYING

La versión del PERFORM VARYING es la más complicada sintácticamente. Se suele emplear cuando se desea ejecutar un conjunto de instrucciones con todos o algunos de los elementos de una tabla de una o más dimensiones.



**FORMATO BÁSICO:**

**PERFORM nombre-párrafo-1 [THRU nombre-párrafo-2]  
 VARYING índice-1 FROM valor-inicial BY +valor-1  
 UNTIL condición-1**

La ejecución del PERFORM VARYING se desarrolla de la siguiente forma:

**PRIMERO:** Se **inicializa** internamente **índice-1** con valor-inicial.

**SEGUNDO:** **EVALUA la condición-1**, si es cierta no se ejecutan las instrucciones comprendidas en nombre-párrafo-1 y se pasa a la instrucción siguiente a PERFORM. Si es falsa **se ejecuta UNA VEZ nombre-párrafo-1**.

**TERCERO:** **Suma o resta** a índice-1, el valor del campo o valor numérico que sigue a BY (valor-1). Y vuelve a analizar la condición-1.

**FIN:** Y el **ciclo se repite** desde el segundo punto hasta que la condición sea cierta.

		A	B															
4	6	7	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	72
010				PERFORM				PROCESAR-ELEMENTOS										
020				VARYING				INDICE-1		FROM	1	BY	+1					
030				UNTIL				INDICE		GREATER	50.							
030																		
010				PERFORM				PROCESAR-ELEMENTOS										
020				VARYING				INDICE-1		FROM	50	BY	-1					
030								UNTIL		INDICE	IS	ZERO.						

**FORMATO COMPLETO: (TABLAS DE MÁS DE UNA DIMENSIÓN).**

```

PERFORM      nombre-párrafo-1  [THRU nombre-párrafo-2]
      VARYING  índice-1  FROM valor-inicial-1 BY  valor-1
      UNTIL    condición-1

      [ AFTER  índice-2  FROM valor-inicial-2 BY valor-2
        UNTIL    condición-2 ]

      [ AFTER  índice-3  FROM valor-inicial-3 BY valor-3
        UNTIL    condición-3 ]

      . . .

```

En este formato la ejecución se lleva a cabo de la siguiente forma:

**PRIMERO:** Se inicia índice-1 con valor-inicial-1 e índice-2 con valor-inicial-2 e índice-3 con el valor-inicial-3.

**SEGUNDO:** Evalúa la **condición-1**, si es cierta, no se ejecuta nada del PERFORM y se salta a la instrucción siguiente al PERFORM VARYING. Si es falsa, se evalúa la condición-2, y si también es falsa, se analiza la condición-3.

**TERCERO:** Mientras la condición-3 es falsa, se ejecutan sucesivamente las declaraciones de nombre-párrafo-1 y se incrementa o decrementa índice-3 con el valor de su BY. Se repite hasta que la condición-3 es cierta.

Cuando la condición-3 es cierta, se vuelve a inicializar índice-3 con valor-inicial-3, se incrementa también índice-2 en el valor de su BY y se pasa al punto anterior. Se repite el proceso anterior hasta que la condición-2 es cierta.

Cuando la condición-2 es cierta, se vuelve a inicializar índice-2 con valor-inicial-2, se incrementa también índice-1 en el valor de su BY y se pasa al punto segundo.

**FIN:** Repitiendo este proceso hasta que la condición-1 se cumple.

Ejemplo:

	A	B
4 6 7 8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72	
010	PERFORM	CALCULO-MEDIA-GASTOS-AÑO-POR-SEMANA
020	VARYING	INDICE-AÑOS FROM 1950 BY 1
030		UNTIL INDICE-AÑOS > 1991
050	AFTER	INDICE-MESES FROM 1 BY 1
060		UNTIL INDICE-MESES > 12
070	AFTER	INDICE-SEMANA FROM 1 BY 1
080		UNTIL INDICE-SEMANA > 4
090	END-PERFORM.	

# ESTRUCTURA DE UN PROGRAMA COBOL

## 1. IDENTIFICACION DEL PROGRAMA (ID DIVISION)

Los datos de identificación de un programa COBOL se dan en la IDENTIFICATION DIVISION (ID DIVISION abreviado).

Es la Primera División con la que se inicia el programa.

### FUNCIONES

\* **DEFINIR** el nombre del PROGRAMA FUENTE Y OBJETO.

\* **INFORMAR** sobre:

- . AUTOR DEL PROGRAMA
- . INSTALACIÓN
- . FECHA ESCRITURA
- . FECHA COMPILACIÓN
- . OBJETIVO DEL PROGRAMA

\* **ESCRIBIR** Los objetivos específicos del programa.

```
0 0    1
7 8...2.....
ID DIVISION.
PROGRAM-ID. nombre-programa.
AUTHOR.    nombre-programador.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
* COMENTARIOS:  xxxxxxxxxxxxxxxx
```

\* Todos los párrafos son opcionales, excepto PROGRAM-ID.

\* Las Sentencias de todos los párrafos, excepto PROGRAM-ID, son descriptivas.  
El compilador los considera comentarios.

\* **Nombre-programa** puede constar hasta de 30 CARACTERES, (sin ningún carácter especial) siendo el primero una letra. El compilador toma sólo los 8 primeros en el caso de que el nombre tuviese más de 8.

\* En DATE-COMPILED la fecha la pone el Compilador, con el formato: MMM.  
DD.AAAA. Ejemplo: DIC.28.1991.

**EJEMPLO DE CODIFICACION DE LA IDENTIFICATION DIVISION:**

			A	B															
4	6	7	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	72	
010			ID DIVISION.																
020																			
030			PROGRAM-ID. FYCXXE01.																
040																			
050			AUTHOR. ESTEBAN DIAZ-MINGO CARREÑO.																
060																			
070			DATE-COMPILED.																
080																			
090			*	Comentario claro y resumido del objetivo del programa.															
100																			
110																			

Información **esencial** a codificar:

**PROGRAM-ID.**

**AUTHOR.**

**DATE-COMPILED.**

**Comentario del objetivo del programa.**

## 2. CARACTERÍSTICAS DE ENTORNO. ENVIRONMENT DIVISION.

### FUNCIONES:

- Definir, asignar, direccionar y especificar peculiaridades de los ficheros que van a ser utilizados en el programa.
- Relacionar los “**nombres lógicos**” de los archivos, que van a ser utilizados por el programa, con los “**nombres reales**” de éstos en el dispositivo de Imacenamiento.
- Especificar opcionalmente, las características y tipo del ordenador que se va a utilizar para “compilar” y “ejecutar” el programa.

ESQUEMA GENERAL																		
			A	B														
4	6	7	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	72
010			ENVIRONMENT DIVISION.															
020			*	=====														
025																		
030			CONFIGURATION SECTION.															
040																		
050			SOURCE-COMPUTER. Comentario															
060																		
070			OBJECT-COMPUTER. Comentario.															
080																		
090			SPECIAL-NAMES.															
100			DECIMAL-POINT IS COMMA.															
110																		
120			INPUT-OUTPUT SECTION.															
020			*	=====														
130			FILE-CONTROL.															
140																		
150			<u>SELECT</u> nombre-fichero-int <u>ASSIGN</u> TO nombre-externo															
190																		
200																		
210			<u>ORGANIZATION</u> IS { INDEXED }															
220			RELATIVE }															
230																		
240			SEQUENTIAL }															
250			<u>ACCESS</u> MODE IS { RANDOM }															
260			DYNAMIC }															
370																		
380			FILE <u>STATUS</u> IS nombre-campo.															
390																		
400																		
410																		

## DESCRIPCIÓN DE SECCIONES: CONFIGURATION SECTION

Descripción de las características del ordenador, a nivel de COMENTARIO.

### Ejemplo:

**SOURCE-COMPUTER. IBM-9000 ó IBM-3090**

Puede describir también el tamaño de memoria que ocupará el programa en palabras, caracteres o módulos. Esta opción no es utilizada en ordenadores de gran capacidad. (Suele referirse a este tipo de ordenadores como entorno "HOST" y/o "MAIN-FRAME").

En esta Sección se pueden reasignar funciones.

**Ejemplo:**

**SPECIAL-NAMES.**

**DECIMAL-POINT IS COMMA.**

En cualquier campo numérico o numérico de edición, la coma hace funciones de punto y el punto de coma.

## **INPUT-OUTPUT SECTION**

Permite enlazar los "**nombres simbólicos**" de los ficheros, (inventado por el programador), con los "**nombres reales**" de los mismos. También, para dar la "**organización**", el "**tipo de acceso**" y **otras** características necesarias, para que todos los archivos puedan ser procesados adecuadamente.

## **PÁRRAFO FILE-CONTROL**

**SELECT nombre-fichero-int ASSIGN TO nombre-externo.**

**SELECT nombre-fichero-int :**

Con la cláusula **SELECT** se definen cada uno de los ficheros del programa.

**ASSIGN TO :**

Con esta cláusula **nombre-fichero-int** se asocia a un **nombre-externo**, que será **el enlace**, que permitirá localizar el dispositivo de almacena mien to que soporta al Fichero Real.

**nombre-externo :**

**Nombre real** del fichero o **nombre de enlace** con él.

(Nombre real en entorno "PC" y nombre de enlace de 1 a 8 caracteres que aparecerá como nombre de la DD en el JCL, si es en el entorno "HOST").



### 3. Data División: Definición de Ficheros y otros Datos

#### INTRODUCCIÓN A LOS DATOS

En una situación de **proceso de información** es imprescindible que **los datos** a consultar o a actualizar estén presentes en la memoria principal del ordenador así como **las instrucciones** necesarias para procesarlos adecuadamente.

**Los datos** para su proceso han de estar perfectamente **localizados**. Su **localización** se realiza en base a una **dirección de comienzo** del dato más la **longitud** predefinida del mismo.

**Dirección y longitud** definen unas posiciones contiguas de memoria donde se alojará un dato. A este conjunto de posiciones contiguas de memoria que almacenarán datos, se les denomina **CAMPOS**.

**En un lenguaje simbólico** no se hace referencia a un direccionamiento real, sino que **se definen nombres asociados a cada uno de los campos**.

En los lenguajes de alto nivel, el Programa Compilador será el que asigne direcciones para cada uno de los campos definidos por el programador. Para referenciar un campo basta con usar el nombre que tiene asociado. La dirección del campo en memoria real es transparente al programador.

En un programa codificado en LENGUAJE COBOL, **los datos** y **los ficheros** en los cuales se almacenan los datos, se definen en la **DATA DIVISION**.

#### DATA DIVISION

Dentro de la DATA DIVISION, **los datos**, se agrupan por SECCIONES, en función del origen o utilidad de esos datos.

Para cada programa se definirán únicamente aquellas secciones que sean necesarias.

#### SECCIONES DE LA DATA DIVISION

##### - FILE SECTION

Para definición de las características de los datos procedentes o con destino a FICHEROS.

### - WORKING-STORAGE SECTION

Para la definición de todos datos intermedios o de trabajo que se consideren necesarios para un adecuado procesamiento de los datos.

### - LINKAGE SECTION

Para definir datos comunes a otro programa con el que se enlaza.

## 4. PROCEDURE DIVISION

### FUNCIÓN

En esta división se **codifican las instrucciones** o sentencias necesarias para el desarrollo de la lógica del PROGRAMA.

Así pues, la procedure estará formada por todas las **instrucciones** que el ordenador deberá “ **ejecutar** “ con los datos de entrada definidos en las secciones anteriores, para obtener los resultados de salida apetecidos.

Las distintas instrucciones se pueden reagrupar, en función del tipo de tratamiento que realizan con los datos, en varios grupos.

Las Instrucciones codificadas en COBOL, a diferencia de otros lenguajes, sirven como AUTODOCUMENTACION del programa ya que tienen la apariencia del lenguaje corriente (aunque en inglés). Esto, sumado a que es un lenguaje fácil de aprender y de mantener, explica porqué el 80% de las aplicaciones que se desarrollan en todo El Planeta, sean en COBOL.

**DISEÑO DE PROGRAMAS DE  
ACTUALIZACIÓN.  
ENFRENTAMIENTO DE  
FICHEROS SECUENCIALES**

## 1.- ENFRENTAMIENTO DE FICHEROS. Actualización de Ficheros Secuenciales.

En la empresa es imprescindible guardar toda la información importante en ficheros. En las tareas habituales de gestión de dicha información, no solamente tenemos que consultar la información guardada en estos ficheros, sino que además, una de las tareas más importantes es la creación y el mantenimiento actualizado al día de estos ficheros.

Si pensamos en un fichero del personal o de clientes o de proveedores o de cualquier otro tipo de información, es imprescindible dar ALTAS (es decir, incluir por ejemplo los nuevos empleados que van siendo contratados en la empresa), dar BAJAS (borrar del fichero de personal aquellas personas, que ya no trabajan en la empresa) y hacer MODIFICACIONES en estos ficheros (cuando una persona cambia de teléfono, de dirección, ha tenido un hijo más, se le ha subido el sueldo, etc., etc).

### 1.1. Operaciones Típicas con los registros de un fichero.

Las operaciones más habituales que debemos realizar con los ficheros son:

- **Altas:** consiste en la adición o inserción de uno o varios registros en el fichero. Esta operación sólo será posible si el fichero ya existe.
- **Bajas:** consiste en eliminar uno o varios registros del fichero. Este proceso requiere un primer proceso de lectura para la localización del registro que se pretende borrar.
- **Modificaciones:** consiste en realizar un cambio total o parcial de uno o varios campos de los registros de un fichero. Primero hay que leer para localizar el registro que se quiere modificar y posteriormente escribir para la actualización de parte o de todo el registro.

- **Consultas:** Nos permite acceder a uno o varios registros con la intención de visualizar el contenido de sus campos en pantalla o impresora en forma de listados ordenados siguiendo criterios de clasificación establecidos por el usuario.

## 1.2. Clasificación de Ficheros.

Podemos hacer múltiples clasificaciones de los ficheros. Vamos a ver algunas de las más universales e importantes:

**1.2.1.- FICHEROS PERMANENTES:** son aquellos cuyos registros sufren pocas alteraciones o variaciones a lo largo del tiempo y contienen información muy valiosa para el buen funcionamiento de la aplicación.

- **Ficheros Históricos:** contienen información acumulada a lo largo del tiempo sobre las actualizaciones sufridas en los ficheros maestros y constantes. Por ejemplo, el fichero de personal de la empresa del que guardaremos una versión al final de cada año, o el inventario al final de año de las existencias de un almacén.

- **Ficheros Maestros o de situación:** son los encargados de mantener constantemente actualizados los campos cuya información es variable. Por ejemplo, el fichero de personal de la empresa que día a día u hora a hora vamos actualizando con las modificaciones que se van dando, o de un fichero de inventario con información sobre la cantidad de piezas existentes en el almacén en cada momento.

- **Ficheros Constantes:** contiene información fija y necesaria para el funcionamiento de la aplicación e información con un bajo índice de variación en el tiempo. Por ejemplo un fichero de códigos postales en el que se relacionen códigos postales con diversas poblaciones y distritos, o el fichero con los códigos de departamentos existentes en la empresa.

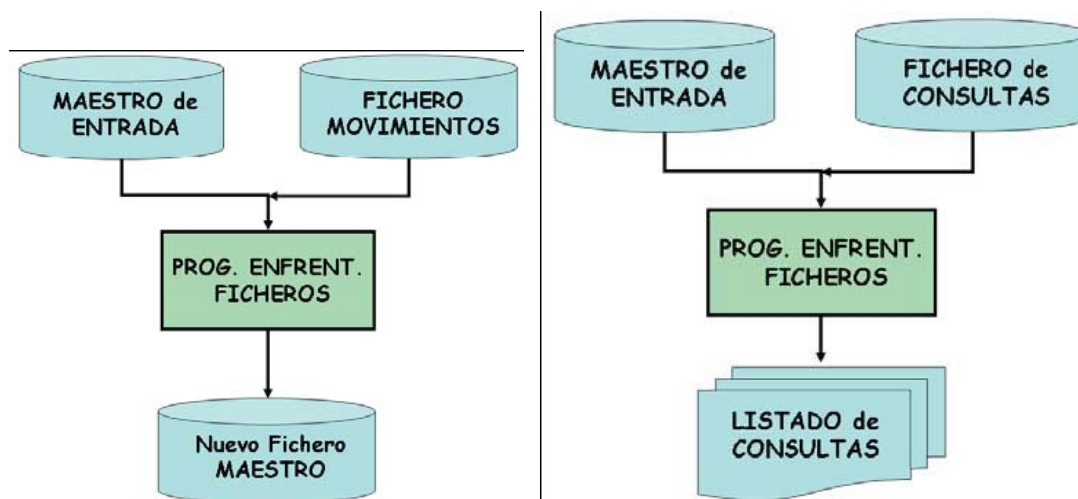
### 1.2.2.- FICHEROS TEMPORALES:

- **Ficheros de movimiento o transacción:** este tipo de ficheros suele contener información necesaria para la actualización de los ficheros maestros. Aquí se debe poner, los registros que se desean dar de alta en los ficheros maestros, los registros que se quieren modificar y a qué campos afecta y los registros que se quieren dar de baja en los ficheros maestros.

- **De maniobra o transitorios:** son verdaderos ficheros auxiliares creados durante la ejecución de programas o aplicaciones con el fin de obtener cierta información que posteriormente será procesada para conseguir unos resultados. Son ficheros que se crean temporalmente para un proceso, una vez que este proceso ha finalizado, este fichero deja de tener valor y se puede borrar del sistema.

### 1.3.- Esquemas básicos de ACTUALIZACIÓN de un fichero MAESTRO a partir de otro fichero de TRANSACCIONES.

Los esquemas más elementales de actualización o de consulta de un fichero maestro en un enfrentamiento son los siguientes:



**Para poder realizar un enfrentamiento de ficheros deben tener las siguientes características:**

**Características de los campos de enfrentamiento:**

- Los registros de **ambos ficheros** secuenciales de entrada **están clasificados** según **el mismo criterio** (por ejemplo en ascendente).
- y **se enfrentan** por **uno o varios campos** de control (que llamaremos **campos de enfrentamiento**).
- Y los campos de enfrentamiento deben ser **el mismo** o los mismos (en caso de ser varios) **para los dos ficheros** a enfrentar.

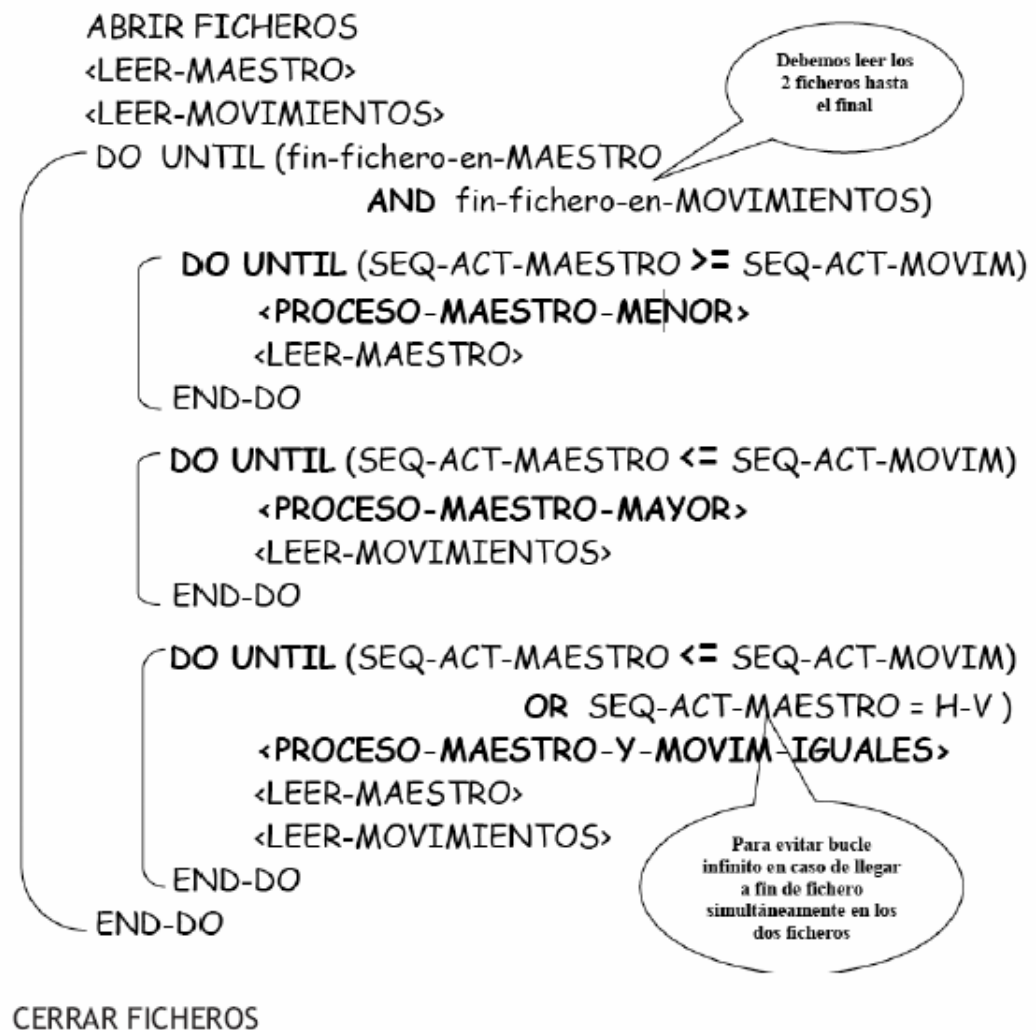
**Situaciones posibles de proceso que se pueden dar en un enfrentamiento:**

Cuando estamos realizando un enfrentamiento de ficheros tendremos que haber leído un registro de cada uno de los ficheros y después se analiza el contenido de los campos claves de los dos ficheros.

A partir de este contenido se pueden dar únicamente las tres situaciones siguientes:

- QUE LA CLAVE DE ENFRENTAMIENTO DEL **MAESTRO SEA MAYOR** QUE LA DE CONSULTAS O MOVIMIENTOS.
- QUE LA CLAVE DE ENFRENTAMIENTO DEL **MAESTRO SEA MENOR** QUE LA DE CONSULTAS O MOVIMIENTOS.
- QUE LA CLAVE DE ENFRENTAMIENTO DEL **MAESTRO SEA IGUAL** QUE LA DE CONSULTAS O MOVIMIENTOS.

Y en función de estas tres situaciones posibles se monta el pseudocódigo básico de un enfrentamiento de ficheros que reflejamos a continuación:

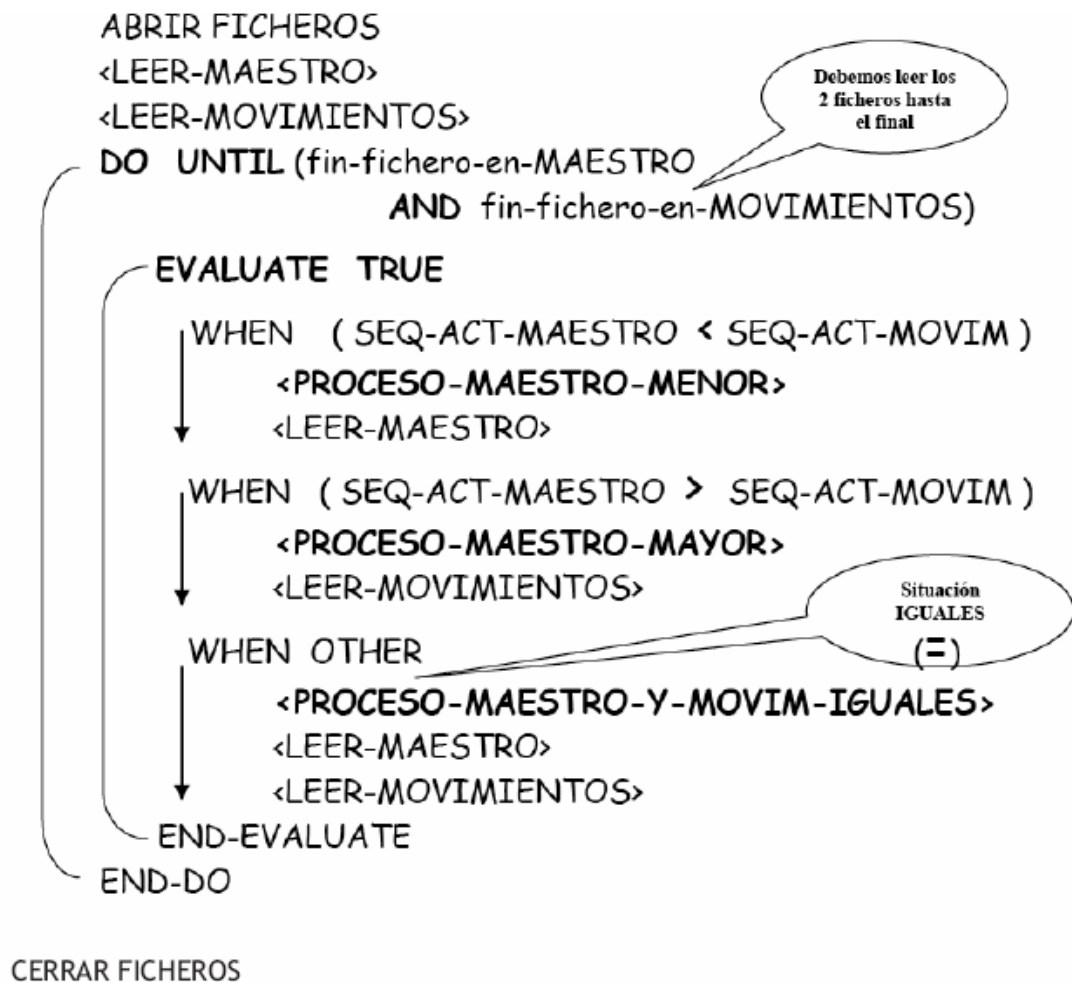
Proceso enfrentamiento (formato1)

En los procesos **maestro-y-movimientos-iguales**, **maestro-mayor**, y **maestro-menor** se pondrán las instrucciones oportunas en función de las particularidades del programa que estemos diseñando, pero el esquema universal válido para montar la línea principal puede ser siempre éste.

Siempre en estas repetitivas se termina leyendo del fichero con clave menor y de los dos si estamos en la situación de iguales.



**Proceso enfrentamiento (formato2) con instrucción CASE del lenguaje COBOL II**



En los procesos **maestro-y-movimientos-iguales**, **maestro-mayor**, y **maestro-menor** se pondrán las instrucciones oportunas en función de las particularidades del programa que estemos diseñando, pero el esquema universal válido para montar la línea principal puede ser siempre éste.

Siempre en estas repetitivas se termina leyendo del fichero con clave menor y de los dos si estamos en la situación de iguales.

# **FUNCIONES Y PROCEDIMIENTOS**

## 1. Teoría y Funciones con Funciones y Procedimientos

### Funciones y Procedimientos

A lo largo éste te mostraremos diferentes cálculos que puedes realizar con los distintos lenguajes de programación

#### - Función de cada módulo

- Realizar una operación independiente de los restantes desde el punto de vista funcional, pero este puede estar relacionado con otros procedimientos y funciones para el intercambio de valores de variables.
- Cualquier algoritmo se puede transformar en un procedimiento para ser utilizado dentro de otro algoritmo mayor.

Parámetros Formales Actuales.



Entre los procedimientos, funciones y su entorno se producen una relación en base a un intercambio de valores de las variables.

## Variables Globales y Locales

La Variable global es aquella que puede ser utilizada (leída, modificada, etc.) a lo largo de todo el algoritmo principal y también por cualquiera de los sub-algoritmos (entiéndase funciones y procedimientos) que componen el algoritmo en sí.

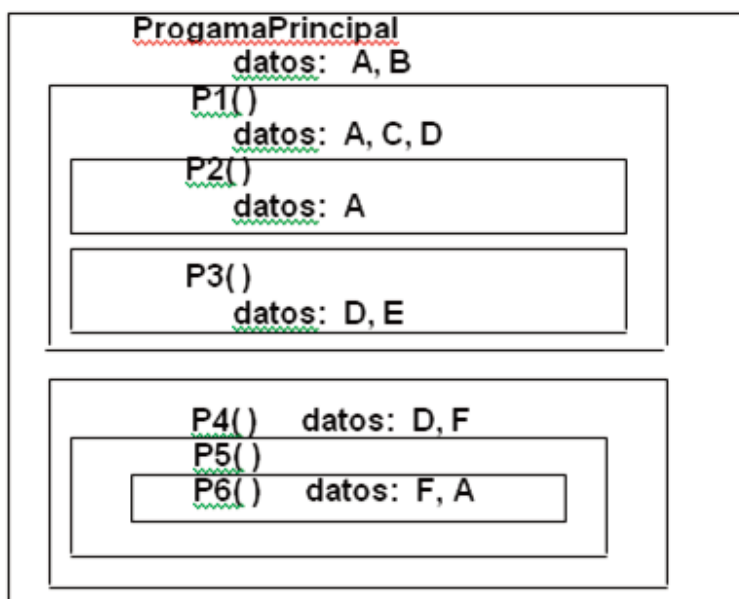
### Variable local

Aquella que sólo puede ser referenciada dentro del sub-algoritmo en el cual ha sido Declarada.

## Funciones y Ámbitos de las Variables

Para ver las funciones y el ámbito de las variables presiona cada botón

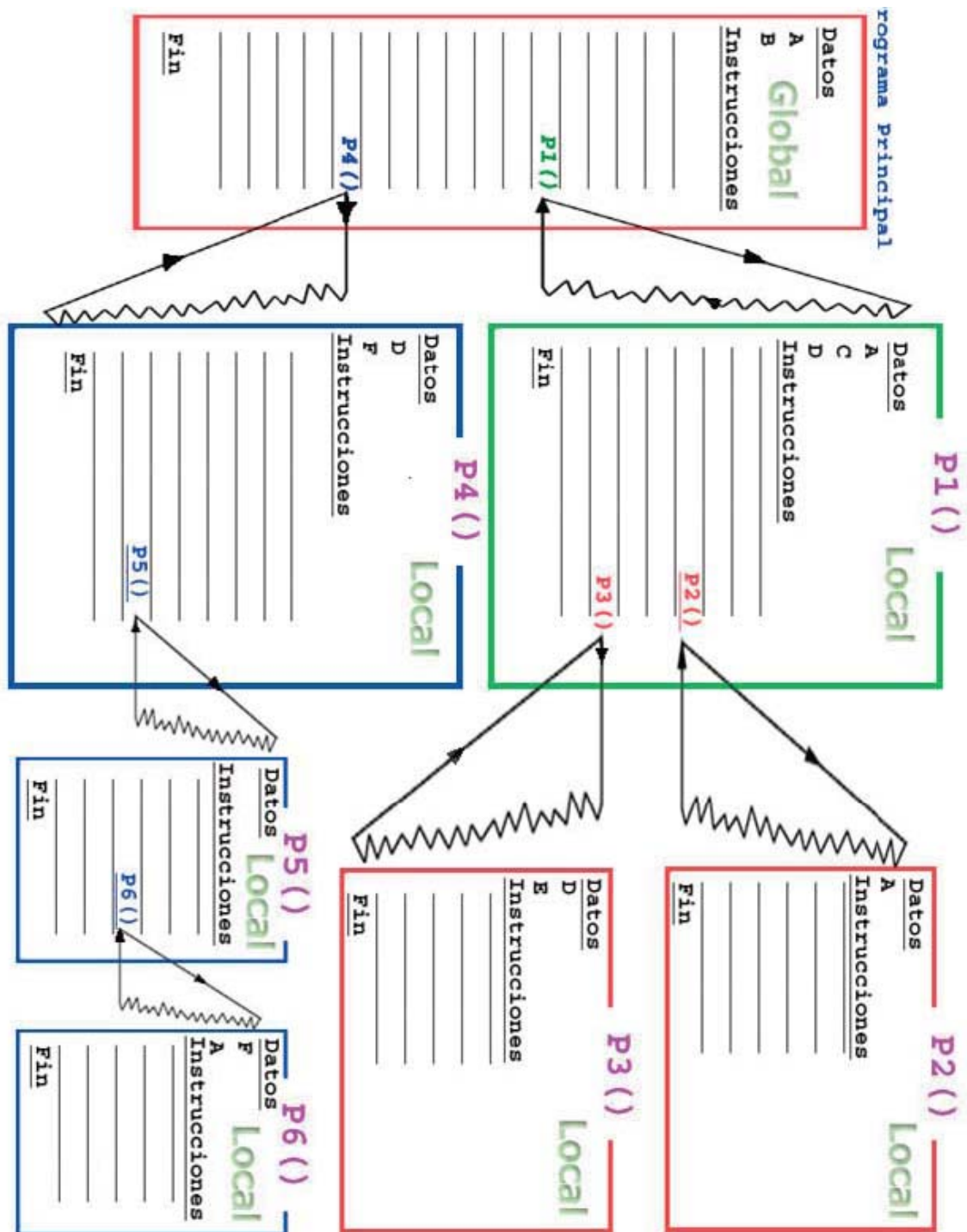
Funciones:



Ámbito de las Variables.

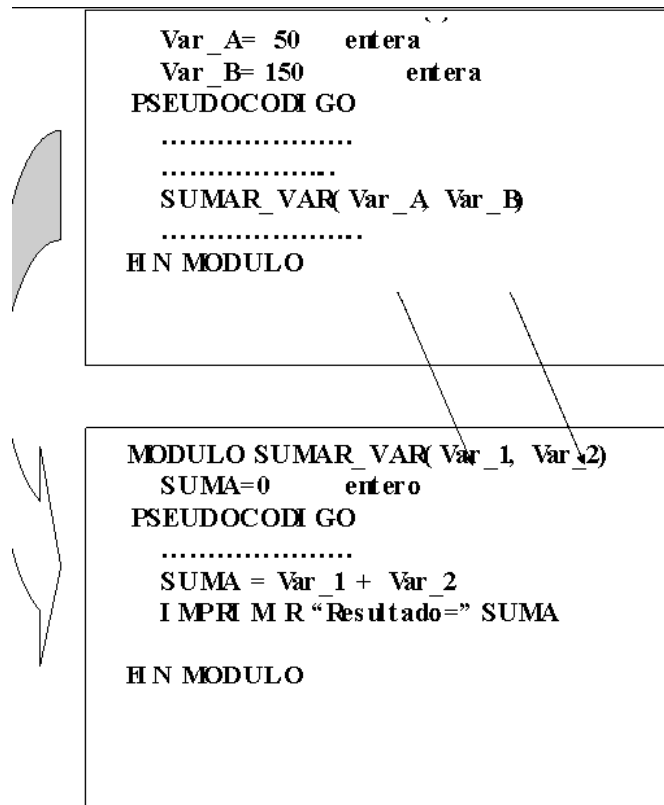
<b>VARIABLES</b>	<b>AMBITO SUBPROGRAMA</b>
A del PP	P4, P5, PP
B del PP	P1, P2, P3, P4, P5, P6, PP
A del P1	P1, P3
C del P1	P1, P2, P3
D del P1	P1, P2
A del P2	P2
D del P3	P3
E del P3	P3
D del P4	P4, P5, P6
F del P4	P4, P5
F del P6	P6
A del P6	P6

## Funciones y Procedimientos. Representación Grafica



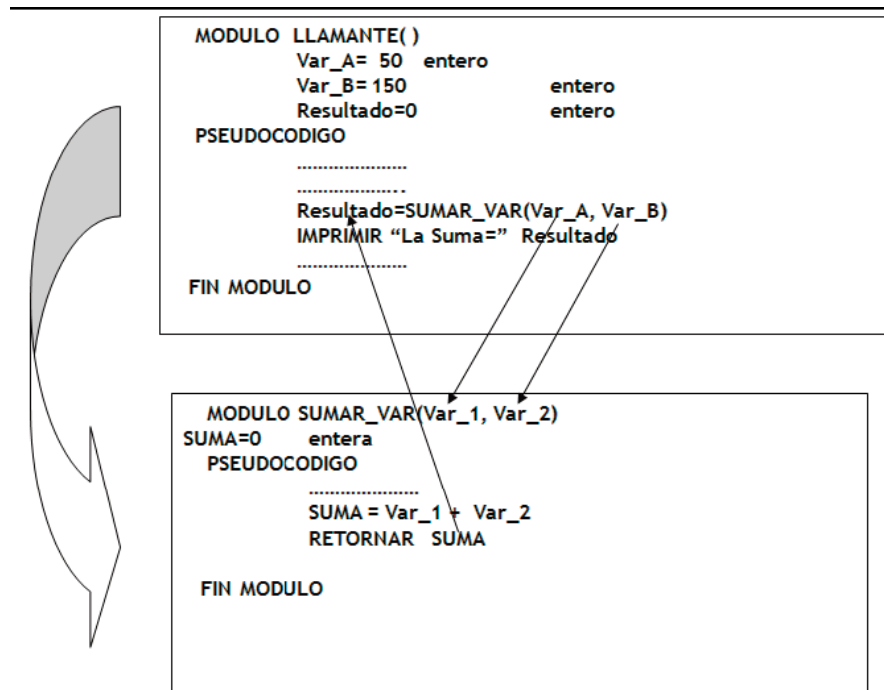
## Llamadas a Procedimientos

A través del esquema siguiente podrá ver como se realizan las llamadas a procedimientos.



## Esquema Básico de Llamadas a Procedimientos

Ahora veremos el esquema básico de llamadas a procedimientos.



### Suma de dos números. Métodos



**METODO 1º)**

Sin necesidad de procedimientos. Todo en el programa principal

**METODO 1º) Sin necesidad de procedimientos.**

Programa: SumaDeDosNumeros

Inicio (Módulo Principal)

DATOS:

Int     A  
Int     B  
Int     Suma

PSEUDOCODIGO

Imprimir "Dame dos Números y los SUMO"

Imprimir "Teclea el primer número"

LEER A

Imprimir "Teclea el segundo número"

LEER B

SUMA = A + B

Imprimir "La SUMA de los Dos números es: " SUMA

Fin

**METODO 2º)**

Usando variables GLOBALES dentro del procedimiento.

Programa: SumaDeDosNumeros

Inicio (Módulo Principal)

DATOS:

Int     A

Int     B

Int     Suma

PSEUDOCODIGO

Imprimir "Dame dos Números y los SUMO"

Imprimir "Teclea el primer número"

LEER A

Imprimir "Teclea el segundo número"

LEER B

Modulo SUMAR\_DOS()

↓  
Fin

Módulo SUMAR\_DOS()

DATOS: (Se usan las variables Globales)

PSEUDOCODIGO:

SUMA = A + B

Imprimir "La SUMA de los Dos números es: " SUMA

↓  
Fin Módulo

**METODO 3º)**

Usando variables LOCALES dentro del procedimiento..

Programa: SumaDeDosNumeros

Inicio (Módulo Principal)

DATOS:

PSEUDOCODIGO

Imprimir "Dame dos Números y los SUMO"

Modulo SUMAR\_DOS()

↓ Fin

Módulo SUMAR\_DOS()

DATOS:

Int A

Int B

Int SUMA

PSEUDOCODIGO:

Imprimir "Teclea el primer número"

LEER A

Imprimir "Teclea el segundo número"

LEER B

SUMA = A + B

Imprimir "La SUMA de los Dos números es: " SUMA

↓ Fin Módulo

**METODO 4º)**

Pasándole los valores de las variables GLOBALES al procedimiento y ejecutándose dentro del mismo con variables LOCALES.

Programa: SumaDeDosNumeros

Inicio (Módulo Principal)

DATOS:

Int A

Int B

PSEUDOCODIGO

Imprimir "Dame dos Números y los SUMO"

Imprimir "Teclea el primer número"

LEER A

Imprimir "Teclea el segundo número"

LEER B

Modulo SUMAR\_DOS( A, B )

↓ Fin

Módulo SUMAR\_DOS( Integer X, Integer Y )

DATOS:

Int SUMA

PSEUDOCODIGO:

SUMA = X + Y

Imprimir "La SUMA de los Dos números es: " SUMA

↓ Fin Módulo

**METODO 5º)**

Pasándole los valores de las variables GLOBALES a la Función y ejecutándose el proceso dentro de la misma con variables LOCALES y una vez ejecutado se devuelve el resultado al módulo principal .

Programa: SumaDeDosNumeros

Inicio (Módulo Principal)

DATOS:

Int     A  
Int     B  
Int     SUMA

PSEUDOCODIGO

Imprimir "Dame dos Números y los SUMO"

Imprimir "Teclea el primer número"

LEER A

Imprimir "Teclea el segundo número"

LEER B

Modulo SUMA = SUMAR\_DOS(A, B )

Imprimir "La SUMA de los Dos números es: " SUMA

Fin

Módulo SUMAR\_DOS( Integer X, Integer Y)

DATOS:  
Integer SUMITA

PSEUDOCODIGO:  
SUMITA = X + Y

RETURN SUMITA	(en C y Java)
Ó	
RETURN X + Y	(en C y Java)
Ó	
SUMAR_DOS = SUMITA	(en Visual Basic)

↓ Fin Módulo

**METODO 6º)**

Pasándole los valores de las variables GLOBALES a la Función y ejecutándose el proceso dentro de la misma con variables LOCALES y una vez ejecutado se devuelve el resultado al módulo principal .

Programa: SumaDeDosNumeros

Inicio (Módulo Principal)

DATOS:

Int     A  
Int     B  
Int     SUMA

PSEUDOCODIGO

Imprimir "Dame dos Números y los SUMO"

Imprimir "Teclea el primer número"

LEER A

Imprimir "Teclea el segundo número"

LEER B

Modulo SUMA = SUMAR\_DOS( &A, &B )

Imprimir "La SUMA de los Dos números es: " SUMA

Fin

Módulo SUMAR\_DOS( Integer \*X, Integer \*Y)

DATOS:

Integer SUMITA

PSEUDOCODIGO:

SUMITA = \*X + \*Y

RETURN SUMITA                   (en C y C++)

Ó

RETURN \*X + \*Y                   (en C y C++)

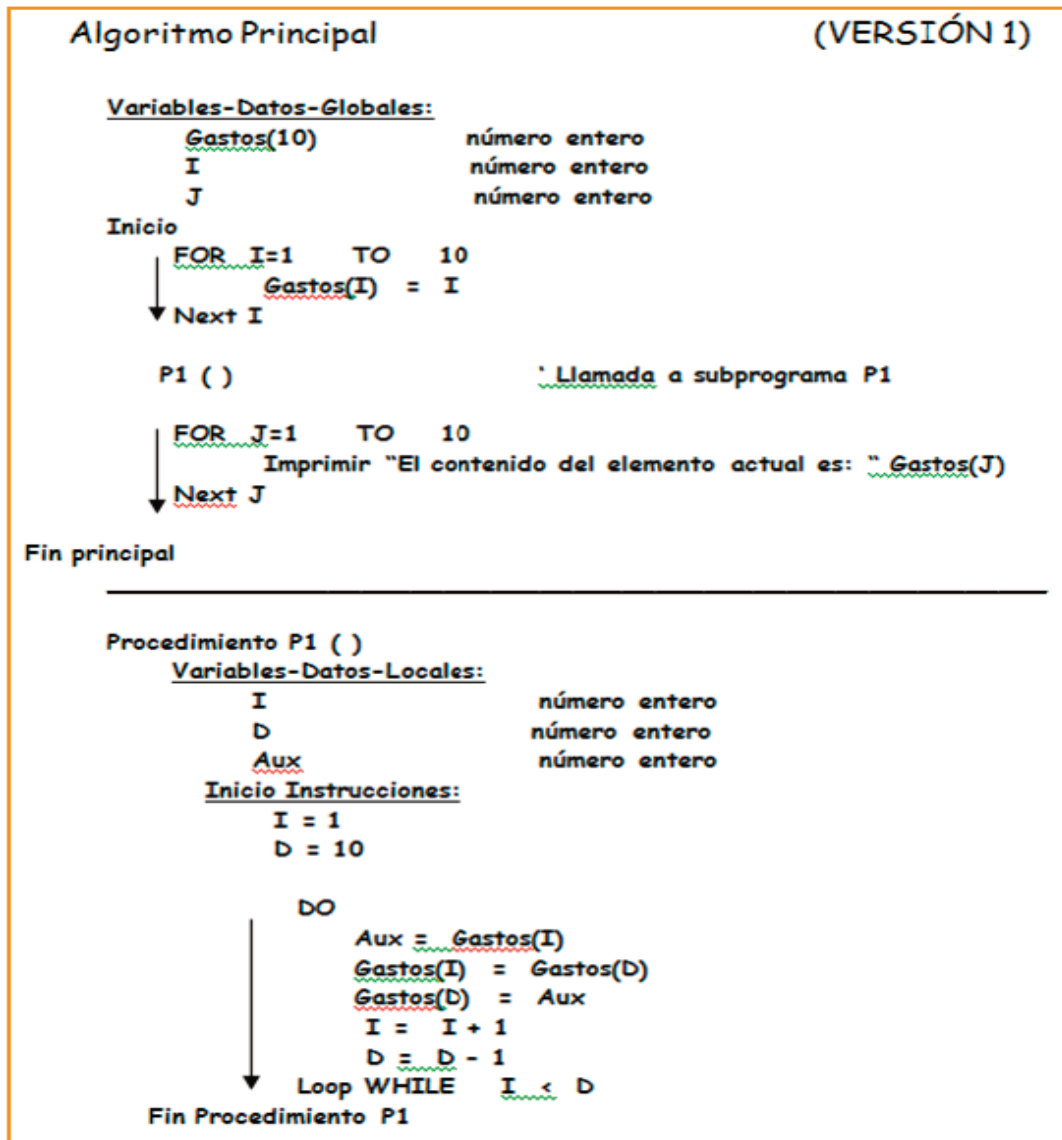
Ó

SUMAR\_DOS = SUMITA           (en Visual Basic)

Fin Módulo

## Anexo Tablas y Procedimiento Versión 1

Estudia el siguiente pseudocódigo y elige las salidas correctas del procedimiento:



### POSIBLES SOLUCIONES:

- a) 2, 4, 6, 8, 10, 12, 14, 16, 18, 20
- b) 4, 6, 12, 16, 21, 27, 34, 42, 51, 61
- c) 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- d) 10, 9, 8, 7, 6, 5, 4, 3, 2, 1



# FUNDAMENTOS DE FICHEROS VSAM Y MANDATOS DEL AMS

## Definición

El IDCAMS es una utilería para gestionar archivos VSAM (Virtual Storage Access Method), aunque también puede ser usada para archivos secuenciales.

Sus funciones principales son:

- Definir y borrar un archivo VSAM
- Copiar de un archivo a otro
- Construir índices alternativos
- Listar catálogos
- Imprimir archivos
- Transferir archivos de un sistema a otro

Un **VSAM** es un método de acceso a los datos almacenados en un dispositivo de acceso directo

Se llaman archivos **VSAM** aquellos que son accedidos de acuerdo con lo descrito anteriormente.

## Tipos de VSAM

Los archivos VSAM pueden clasificarse de la siguiente forma:

### ESDS

Los registros son almacenados y recuperados en el mismo orden en el que se graban

### KSDS

Los registros se recuperan en el orden de un campo definido como clave. Son los más usados.

### RRDS

Los registros se almacenan en el orden dado por un número de secuencia.

### **Archivos ESDS**

Los archivos ESDS (Entry Sequenced Data Storage) tienen las siguientes características:

- No tienen índices ni claves
- Los registros se añaden al final del archivo
- Sólo pueden ser accedidos en forma secuencial
- Son similares a los archivos secuenciales, pero no pueden grabarse en cinta

### **Archivos KSDS**

Los archivos KSDS (Key Sequenced Data Storage) tienen las siguientes características:

- Necesitan una clave, que debe ser única, para identificar el registro.
- La clave se utiliza para insertar el registro en el archivo y recuperarlo después.
- Admite claves alternativas (secundarias), no únicas
- Admite acceso directo (por clave) o secuencial

### **Archivos RRDS**

Los archivos RRDS (Relative Record Data Storage) tienen las siguientes características:

- No tienen índices ni claves.
- Los registros se añaden según un número asignado de 1 a n, siendo n el número máximo de registros del archivo.
- Este número puede ser asignado por el programa del usuario o por el sistema.
- Admite acceso directo (por número) o secuencial.

**Codificación**

```
//stepname EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
...
...
/*
```

- Debe codificarse después de la sentencia EXEC como PGM=IDCAMS
- En SYSPRINT se encuentran los mensajes de salida del programa
- SYSIN muestra la codificación de los mandatos

**Creación de un archivo VSAM**

```
//DEFINIR EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER(NAME(ACCEN.MASTER.EMPL)-
    REC(500 50)-
    RECSZ(25 50)-
    KEYS(10 0)-
    VOLUME(DIR003)-
    SHR(2 3)-
    INDEXED -
DATA(NAME(ACCEN.MASTER.EMPL.DATA) -
INDEX(NAME(ACCEN.MASTER.EMPL.INDEX))
/*
```

En algunas instalaciones la definición de archivos **VSAM** es realizada por el departamento de administración de bases de datos

- El archivo es definido como un Cluster (**DEFINE CLUSTER**)
- Un **CLUSTER** en archivos **KSDS**, se encuentra formado por dos componentes: uno de los índices (**INDEX**) y otro los datos (**DATA**)

- Los demás tipos de archivos sólo están formados por datos (**DATA**)
- El parámetro **NAME** identifica al archivo en el catálogo.
- El parámetro **REC** indica el espacio requerido, el cual puede ser especificado en número de registros, tracks ó cilindros.
- El primer subparámetro indica la extensión primaria al crear el archivo y el segundo subparámetro indica la extensión secundaria a incrementar.
- El parámetro **RECZ** indica el tamaño del registro.
- El primer subparámetro indica el tamaño medio en bytes de los registros y el segundo subparámetro indica el tamaño máximo en bytes de los registros.
- **KEYS** define la clave de acceso al archivo.
- El primer subparámetro indica el tamaño en bytes de la clave y el segundo subparámetro la posición relativa a partir del comienzo del registro.
- La primera posición se codifica como 0.
- El parámetro **VOLUME** especifica el nombre del disco en el que se va a generar el archivo.
- El parámetro **SHR** indica las opciones activas, en caso de que el archivo se encuentre compartido.
- El primer sub-parámetro indica el nivel de acceso al archivo por una sola computadora y el segundo se refiere al nivel de acceso en caso de sean varias computadoras conectadas a través de una red.
- El parámetro **INDEXED** indica que el archivo a crear es un **KSDS**.
- Si este parámetro no es codificado se asume **INDEXED**.

#### **DATA(NAME(ACCEN.MASTER.EMPL.DATA**

- Indica el nombre del archivo donde se almacenaran los datos del archivo.

#### **INDEX(NAME(ACCEN.MASTER.EMPL.INDEX))**

- Este parámetro se utiliza para definir el nombre del índice del archivo.

**Borrado de un archivo VSAM**

```
//BORRADO EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
        DELETE ACCEN.MASTER.EMPL
```

- El parámetro **DELETE** permite borrar archivos VSAM.
- Al borrar el **CLUSTER** se eliminan también el **DATA** y el **INDEX** que se encuentran asociados al archivo.
- Esta sintaxis también puede ser usada para el borrado de archivos secuenciales o particionados.

**Copia de un archivo VSAM**

```
//DEFINIR EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//file1 DD DSN=...
//file2 DD DSN=...
//SYSIN DD *
        REPRO INFILE(file1) -
        OUTFILE(file2) -
        SKIP(nnn) -
        COUNT(nnnn)
/*
```

- El parámetro **REPRO** permite:
  - \* Copiar de VSAM a VSAM
  - \* Copiar de secuencial a secuencial
  - \* Convertir de secuencial a VSAM
- Los parámetros **INFILE** y **OUTFILE** especifican los archivos de entrada /salida.
- El parámetro **SKIP** es opcional y es usado cuando se quiere indicar el número de registros que deben “saltarse” antes de comenzar la copia.
- Otra opción es utilizar en el caso de estos archivos el parámetro **FROMKEY**, el cual indica la clave inicial a partir de la cual se quiere copiar.

**FROMKEY ('LOPEZ ')**

- El parámetro **COUNT** indica el número de registros que quieren copiarse.
- En vez de este parámetro puede usarse **TOKEY**, en el cual se puede indicar la clave final.

**TOKEY ('LOPEZ99999')****Impresión de un archivo VSAM**

```
//DEFINIR EXEC PGM=IDCAMS
```

```
//SYSPRINT DD SYSOUT=*
```

```
//namefile DD DSN=...
```

```
//SYSIN DD *
```

```
    PRINT INFILE(namefile) -
```

```
    FROMKEY('LOPEZ ')
```

```
    CHAR
```

```
/*
```

- El parámetro **PRINT** permite imprimir tanto archivos VSAM como secuenciales.
- Para limitar el listado se utilizan los mismos parámetros que en **REPRO**.
- El formato del listado puede ser de tres tipos:
  - \* CHAR Caracteres
  - \* HEX Hexadecimal
  - \* DUMP character-Hexadecimal

# **INSTRUCCIONES DE ENTRADA/SALIDA DE FICHEROS SECUENCIALES**



## 1. INSTRUCCIONES DE ENTRADA/SALIDA

### FUNCIÓN:

Poner en comunicación al PROGRAMA con los DISPOSITIVOS EXTERNOS donde están o estarán almacenados los datos que se desean procesar.

### TIPOS DE INSTRUCCIONES DE ENTRADA/SALIDA

Las instrucciones de entrada salida son las siguientes:

### MISIÓN VERBO

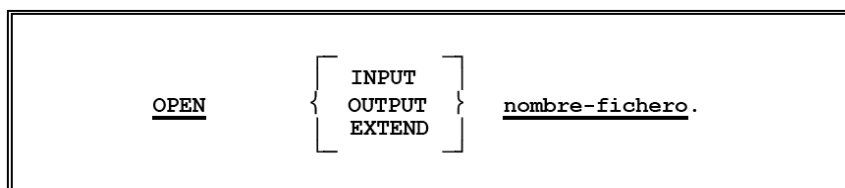
* ABRIR FICHEROS	>	OPEN
* CERRAR FICHEROS	>	CLOSE
* LEER FICHEROS	>	READ
* ESCRIBIR REGISTROS	>	WRITE
* ACEPTAR DATOS	>	ACCEPT
* VISUALIZAR DATOS	>	DISPLAY

## 2. VERBO OPEN

### FUNCIÓN:

Facilita la apertura de los Ficheros, para que posteriormente puedan ser utilizados en operaciones de Lectura, de Escritura o ambas a la vez.

### FORMATO:



**OPEN:** Verbo de la sentencia.

**INPUT:** Fichero abierto para leer de él.

**OUTPUT:** Fichero abierto para grabar en él.

**EXTEND:** Fichero abierto para grabar en él, a partir del último registro. (El contenido inicial del fichero se mantiene).

**nombre-fichero:** Nombre del FICHERO definido en la ENVIRONMENT DIVISION en el lugar de nombre-interno.

**NOTAS:** Cuando se procesa esta sentencia, se verifican las etiquetas de cabecera del fichero (si INPUT) o se comienzan a grabar (si OUTPUT). Con una sola sentencia se pueden abrir varios ficheros.

Un fichero NO se puede abrir, si previamente no está cerrado.

### 3. VERBO CLOSE

#### **FUNCIÓN:**

Cerrar los ficheros previamente abiertos y realizar funciones específicas como generar etiquetas de FIN de FICHERO.

#### **FORMATO:**

**CLOSE nombre-fichero1 [, nombre-fichero2].**

**CLOSE:** Verbo de la sentencia.

**nombre-fichero1:** Nombre de Fichero definido en la ENVIRONMENT DIVISION.

Se pueden 'cerrar' varios ficheros en la misma sentencia.

#### 4. VERBO READ

##### **FUNCIÓN:**

Cada vez que desde un Programa se da una orden de LECTURA, se DISPONE de un REGISTRO LÓGICO para su tratamiento.

**Existen dos formatos :**

1 - LECTURA SECUENCIAL.

2 - LECTURA AL AZAR. (Se tratará más adelante).

##### **LECTURA SECUENCIAL**

Los registros se recuperan SECUENCIALMENTE. Se procesan todos los registros del fichero, uno detrás de otro, hasta que no queda ningún registro. Hasta que se llega al fin del fichero.

Esta situación, se detecta, ya que, se realiza una LECTURA de más, que no recupera NINGÚN REGISTRO, y entonces, el Sistema Operativo facilita a nuestro programa la INDICACIÓN de que se está ya al FINAL DEL FICHERO.

##### **FORMATO DE LECTURA SECUENCIAL**

**READ nombre-fichero [ INTO n-registro-W ]**

**AT END sentencia-imperativa-1**

**[ NOT AT END sentencia-imperativa-2 ].**

**READ:** Verbo de la SENTENCIA.

**nombre-fichero:** NOMBRE del fichero interno definido en la ENVIRONMENT DIVISION.

**INTO n-registro-W:** Opción para disponer del registro también en una zona de trabajo.

**AT END:** Condición que será verdadera, solamente cuando se LEA y NO existan ya REGISTROS que recuperar.

**Sentencia-imperativa-1:** Será el tipo de acción a tomar cuando se ha **terminado** de leer todos los registros del FICHERO.

**NOT AT END: Condición** que será verdadera cada vez que se LEA y **se haya recuperado** un registro. Es decir, siempre que no se detecte el fin del fichero.

**Sentencias-imperativa-2:** Será el tipo de acción a tomar si se recuperó un registro. (De la versión revisada del COBOL-85).

### **ACLARACIONES AL RESPECTO.**

Recuerda que:

- **No** se podrá ejecutar una sentencia READ si previamente no se abrió el fichero (OPEN).
- Sólo se pueden ejecutar sentencias de lectura, (READ) en ficheros abiertos para entrada de datos hacia el programa (INPUT).
- No se podrá ejecutar una sentencia READ una vez detectado el fin del fichero.
- La cláusula FILE STATUS puede sustituir a la opción AT END y NOT AT END. El valor "00" indica registro recuperado, el valor "10" indica Fin de Fichero.
- En realidad los registros lógicos se recuperan agrupados en bloques. Es decir, en registros físicos. El programa, sin embargo, sólo dispone de UN SOLO REGISTRO LÓGICO, cada vez que se ejecuta una sentencia READ.

Ejemplo:

	A	B
4 6 7 8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72	
010	MOVE	SEG-ACTUAL TO SEQ-ANTERIOR.
020		
030	READ	FI-SECUENCIAL INTO REG-TRABAJO
040		AT END
050		MOVE HIGH-VALUES TO SEQ-ACT
060		
070		NOT AT END
080		MOVE SEQ-ENT TO SEQ-ACT
090		ADD 1 TO CONTA-LEIDOS
100		PERFORM COMPROBAR-ORDENACION
120		END-READ.

## 5. VERBO WRITE

### FUNCIÓN:

Posibilitar la grabación o impresión de un registro lógico en un dispositivo de salida.

### EXISTEN DOS FORMATOS LÓGICOS DE ESCRITURA:

- FORMATO PARA **GRABAR** EN SOPORTES MAGNÉTICOS (DISCOS).
- FORMATO PARA **IMPRIMIR** USANDO LA IMPRESORA (PAPEL).

#### 5.1 FORMATO PARA GRABAR EN SOPORTES MAGNÉTICOS

**WRITE n-registro [ FROM n-registro-W ]**

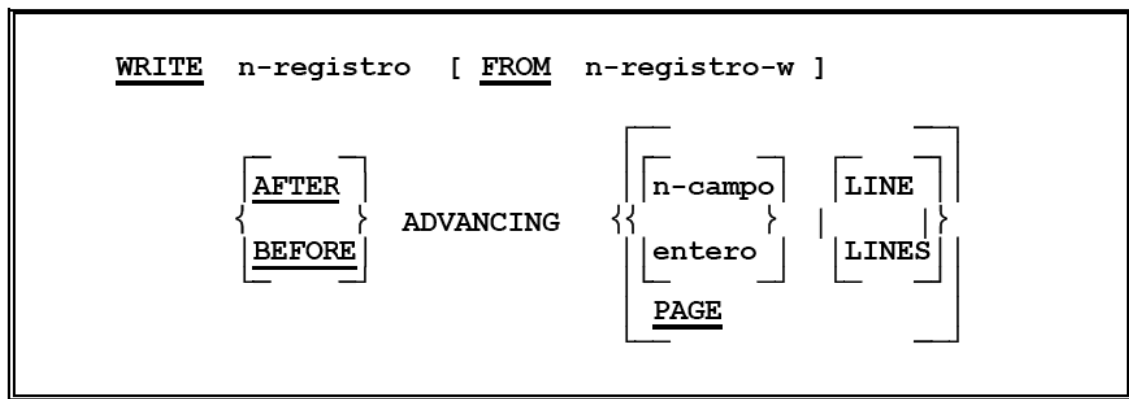
Cada nuevo registro, se grabará, justamente detrás del anterior.

#### 5.2 FORMATOS DE IMPRESIÓN

Estos formatos se utilizan para facilitar la IMPRESION de registros (líneas), los cuales, posteriormente van destinados a PAPEL.

Existen bastantes opciones de impresión. A continuación, se describen las más frecuentes.

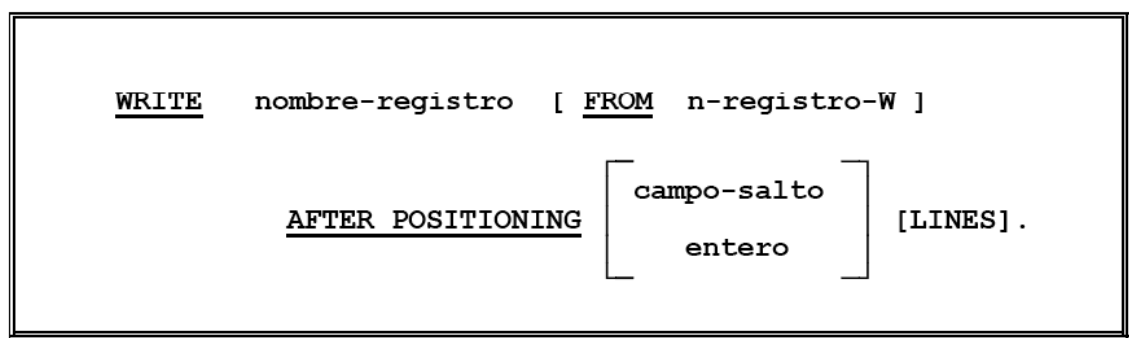
### 5.3 OPCIÓN WRITE ... AFTER ADVANCING



**AFTER:** Escribir el registro, pero **antes de imprimir saltar** una página, o las líneas que indique n-campo o el valor numérico entero.

**BEFORE:** Escribir el registro, pero **después de imprimir salta** el número de líneas que se indique o comenzar en página nueva, en función de lo que se indique.

### 5.4 OPCIÓN WRITE ... AFTER POSITIONING



**WRITE:** Verbo de la sentencia.

**Nombre-registro:** Definido en la FD correspondiente.

**FROM:** Como en Formatos anteriores.

**AFTER POSITIONING:** Imprimir **después de posicionarse** el carro de la impresora.

**Campo-salto ó entero:** El carro de la impresora se posicionará después de averiguar el número de líneas que ha de saltar antes de imprimir. Esto se indica en nombre-campo o mediante un literal numérico entero.

Nombre-campo también es conocido como “campo-salto”.

\* Los distintos contenidos y significados que puede soportar **campo-salto** son:

**b** (blancos) Espaciado sencillo.

**0** (cero) Doble espaciado.

**-** (guión) Triple espaciado.

**+** (más) NO espaciado.

**1-9** Salto a canal 1 a 9.

**A,B,C** Salto a canal 10 a 12.

\* Posibles valores y significados de **entero**:

**0** Salto a canal 1.

**1** Espaciado sencillo.

**2** Doble espaciado.

**3** Triple espaciado.

**NOTAS:** Será posible codificar cualquiera de las opciones POSITIONING, o ADVANCING en función del compilador y del entorno donde se vayan a ejecutar los programas.

En un mismo programa, **no esta permitido mezclar los dos formatos** de impresión.

# INSTRUCCIONES DE ORGANIZACIÓN Y CONTROL



## 1. INSTRUCCIONES DE ORGANIZACIÓN Y CONTROL

### FUNCIÓN:

Se agrupan aquí, aquellos VERBOS, que FACILITAN AJUSTAR LAS DECLARACIONES COBOL a la LÓGICA ESPECÍFICA de un PROGRAMA CONCRETO.

Estos verbos **permiten variar el orden de ejecución SECUENCIAL de las instrucciones (por el ordenador);**

pudiéndose transferir el control a cualquier parte del programa, que el programador considere necesario. **Los verbos que permiten controlar la ejecución secuencial de instrucciones son:**

- IF
- PERFORM
- EVALUATE
- STOP RUN

### Función específica de cada verbo:

**IF** : Se emplea para el desarrollo de ESTRUCTURAS de tipo **alternativo**.

**PERFORM** : Este verbo, provoca un desvío o **salto con retorno** a otro punto del Programa.

Se usa para el **desarrollo de rutinas**. Es decir para agrupar funciones. (INVOCACIONES de Pseudocódigo). También, para desarrollar ESTRUCTURAS de tipo **repetitivo**.

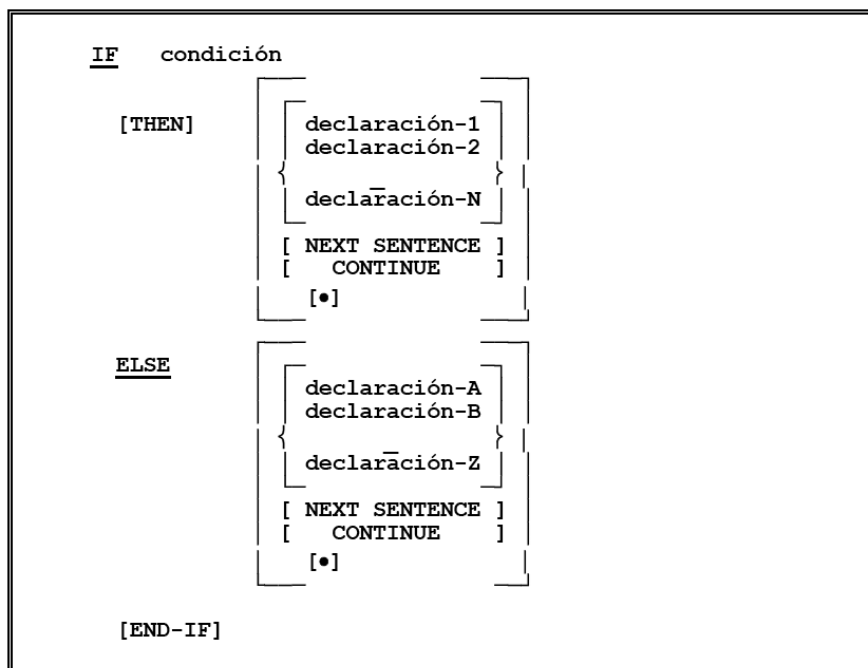
**EVALUATE** : Implementación de la estructura DO CASE del pseudocódigo. Facilita la codificación de **alternativas múltiples**, evitándonos múltiples IF anidados.

**STOP RUN:** Indica FIN de programa. El programa deja de ejecutarse y se devuelve el Control nuevamente al Sistema Operativo.

## 2. VERBO IF

El verbo IF permite analizar una condición, y en función del resultado obtenido, ejecutar **selectivamente** una o más declaraciones. Permite alterar el orden de ejecución secuencial.

FORMATO:



**El formato del IF se lee o interpreta como sigue:**

Si (**IF**) se cumple la condición entonces (**THEN**) ejecútense las declaraciones que siguen a THEN ( y sáltense las declaraciones que siguen al ELSE), sino (**ELSE**), ejecútense las declaraciones que siguen al ELSE.

El punto “.” delimita el alcance de la sentencia y a partir de la revisión del 85 es posible usar la palabra **ENDIF**.

**CONTINUE** ordena saltar a la siguiente instrucción después del **IF**. **NEXT SENTENCE** obliga a saltar a la siguiente sentencia, es decir, saltar TODO hasta el próximo punto. Su uso, facilita la codificación de las condiciones en afirmativo o negativo, según convenga. Estas cláusulas se codificarían, si se considerara oportuno, en la parte del IF que no contenga ninguna declaración.

Si se codifican varios **IF ANIDADOS**, sólo debe de haber **un punto al final** de todos ellos.

#### Ejemplos:

	A	B																	
	4	6	7	8	12	16	20	24	28	32	36	40	44	48	52	56	60	68	72
010			(1)		IF	EDAD-E			>	25									
020						ADD		INCREMENTO				TO		SALARIO-ANTIGUO					
030						MOVE		LIN-DETALLE				TO		LIN-SALIDA					
040						PERFORM		IMPRIMIR-EN-ACTUALIZACION•											
040																			
.....																			
010			(2)		IF	SEQ-ACT		=	HIGH-VALUES										
020						MOVE		'FIN'				TO		CAMPO-FIN					
030						DISPLAY		'** FIN PROGRAMA **'											
040						PERFORM		RUTINA-FIN-PROGRAMA											
050					ELSE														
060						MOVE		REG-E				TO		REG-W					
070						ADD		1				TO		CONT-LEIDOS					
080						PERFORM		PROCESAR-UN-REGISTRO•											
080																			
090					PERFORM		OTROS-TRATAMIENTOS-POSIBLES.												

## EXPLICACIÓN DE LOS EJEMPLOS:

**(1)** Si la respuesta es SÍ; es decir si se cumple la CONDICIÓN, se ejecutan las declaraciones que siguen de inmediato a la CONDICIÓN **hasta el primer punto**. (Parte THEN DEL IF).

En caso contrario, se ejecutan únicamente las declaraciones que siguen al primer punto. (Parte ELSE del IF).

**(2)** Si la respuesta es SI, es decir si se cumple la condición, se ejecutan las declaraciones que están ANTES del ELSE. En caso contrario se ejecutan las declaraciones que siguen al ELSE hasta el **punto primero**. En cualquiera de los 2 casos seguirá después de este punto.

## OPCIÓN END-IF

La implementación del **END-IF**, de la versión del 85, resuelve la “debilidad” de la versión anterior, ya que el uso de este recurso, facilita considerable mente la codificación de IF ANIDADOS; haciendo posible transcribir “literalmente” el diseño del programa reflejado por el pseudocódigo al lenguaje COBOL. El ámbito de cada IF acaba en su **END-IF** correspondiente.

Ejemplo:

	A	B
4 6 7	8	12 16 20 24 28 32 36 40 44 48 52 56 60 68 72
010		
015		MAESTRO-MAYOR.
020		IF CLAVE = 'I'
030		PERFORM 100-INCLUIR
040		ELSE
050		IF CLAVE = 'M'
060		MOVE ERROR-M TO TIPO-ERROR
070		ELSE
080		IF CLAVE = 'S'
090		MOVE ERROR-S TO TIPO-ERROR
100		ELSE
110		MOVE CLV-ERRONEA TO TIPO-ERROR
120		END-IF
128		PERFORM RUTINA-XX
130		END-IF
140		PERFORM LISTAR-ERROR
150		END-IF•
160		PERFORM LEER-MOVIMIENTOS.

## 2.1 CONDICIONES SIMPLES Y TIPOS DE COMPARACIONES.

Los operadores de las condiciones simples pueden ser:

- De RELACION
- De SIGNO
- Y de CLASE

(Los cuales se correlacionan con las condiciones de relación, de signo y de clase, respectivamente).

## 2.2 CONDICIONES DE RELACIÓN

Comprueba si el operando-1 es mayor, igual o menor que el operando-2.

### FORMATO SIMPLIFICADO:

**IF operando-1 OPERADOR operando-2**

Como operandos pueden intervenir:

- nombres de campos
- Literales de cualquier tipo.
- Constantes figurativas
- Expresiones aritméticas.

El operador puede ser:

- |                |     |             |
|----------------|-----|-------------|
| - GREATER THAN | ó > | (mayor que) |
| - LESS THAN    | ó < | (menor que) |
| - EQUAL TO     | ó = | (igual a)   |

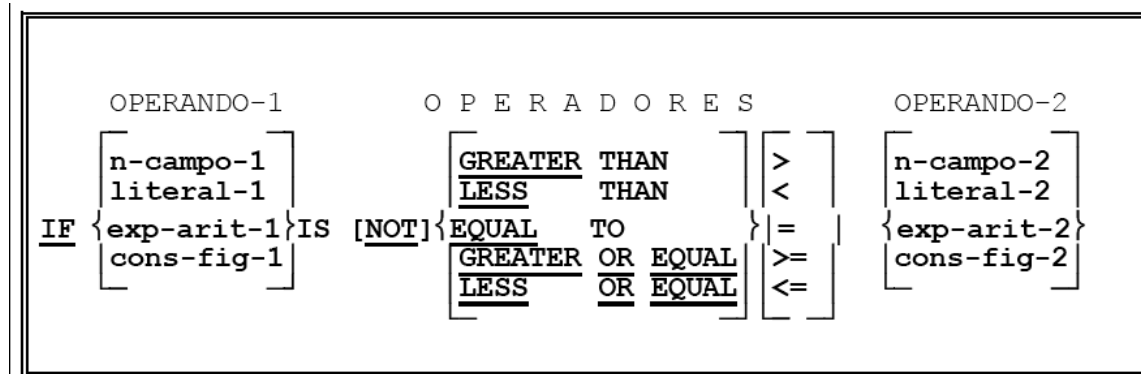
y sus contrarios:

- |                    |         |                |
|--------------------|---------|----------------|
| - NOT GREATER THAN | ó NOT > | (no mayor que) |
| - NOT LESS THAN    | ó NOT < | (no menor que) |
| - NOT EQUAL TO     | ó NOT = | (no igual a)   |

en COBOL II además:

- |                            |      |                 |
|----------------------------|------|-----------------|
| - GREATER THAN OR EQUAL TO | ó >= | (mayor o igual) |
| - LESS THAN OR EQUAL TO    | ó <= | (menor o igual) |

### 2.3 FORMATO COMPLETO PARA CONDICIONES RELACIONALES:



El tipo de comparación que realizará el ordenador dependerá de la forma en que estén definidos los operandos a comparar.

Si ambos son numéricos, la comparación será numérica.

Si son alfanuméricos, la comparación será alfanumérica.

### 2.4 COMPARACIONES NUMÉRICAS

Se realizan algebraicamente de IZQUIERDA a DERECHA. Se igualan internamente las longitudes de los dos operandos. Cuando se encuentra un número que esté en contra de lo preguntado, es decir, cuando se confirma que la condición **NO** se cumple, se para la comparación y se pasa a ejecutar el bloque **ELSE**. Si se llega al final de la comparación, y la condición se cumple, se ejecuta el bloque **THEN**. Los operandos sin signo son considerados positivos.

### 2.5 COMPARACIONES NO NUMÉRICAS

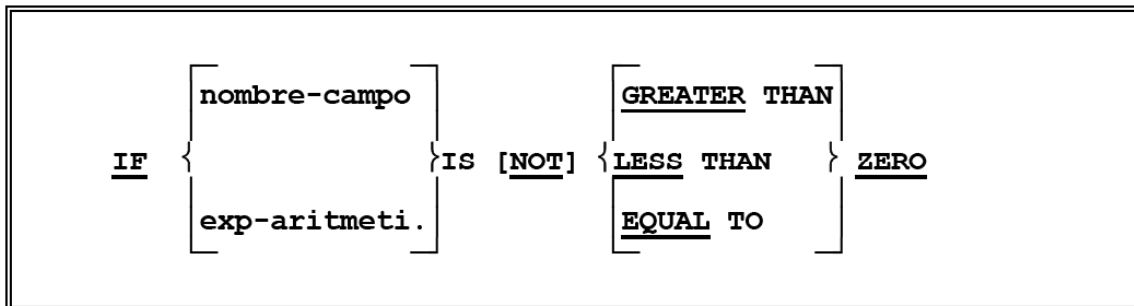
Se realizan de IZQUIERDA a DERECHA carácter a carácter. Si son de longitudes distintas, el campo más pequeño es expandido internamente en un campo de trabajo que se crea el ordenador y se rellena con blancos.

La comparación se realiza, ateniéndose al orden de jerarquía (mayor/menor) preestablecido por el CÓDIGO usado por el ordenador para la representación interna de los caracteres (EBCDIC ó ASCII).

## 2.6 CONDICIONES DE SIGNO

Comprueba si el operando-1 es positivo, cero o negativo. El operando-1 será un campo o una expresión aritmética.

Es un caso particular de las condiciones de relación en el que el segundo operando es **cero**.

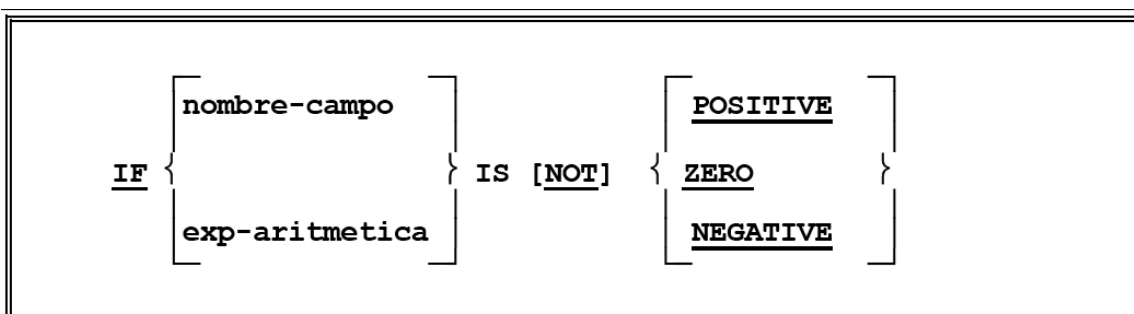


El COBOL permite **simplificar** este tipo de condiciones, dado que:

- GREATER THAN **0** es lo mismo que POSITIVE
- LESS THAN **0** es lo mismo que NEGATIVE
- EQUAL TO **0** es lo mismo que ZERO

\* Por tanto, puede suprimirse el operador, quedando, en definitiva, la condición como sigue:

**FORMATO-2 :**



## 2.7 CONDICIONES DE CLASE

Comprueba si el operando-1 es ALFABÉTICO (A a Z y espacio) o NUMÉRICO (caracteres numéricos de 0 al 9 además del signo apropiado). El operando-1 será un nombre de campo.

Es un caso particular de las condiciones de relación en que el operador es EQUAL TO y el operando-2 es ALPHABETIC o NUMERIC.

Pero como EQUAL TO ALPHABETIC es lo mismo que ALPHABETIC, y EQUAL TO NUMERIC es lo mismo que NUMERIC, se suprime el operador y en definitiva la condición, se codificará como sigue:

#### FORMATO :

<u>IF</u> nombre-campo	IS	[ <u>NOT</u> ]	{	NUMERIC	}
				ALPHABETIC	
				ALPHABETIC-LOWER	
				ALPHABETIC-UPPER	

NUMERIC: Para campos numéricos y alfanuméricos.

ALPHABETIC: Para campos alfabéticos y alfanuméricos.

ALPHABETIC-LOWER: Para detectar si nombre-campo tiene un contenido alfabético en minúsculas.

ALPHABETIC-UPPER: Para detectar si nombre-campo tiene su contenido alfabético en mayúsculas.

## 2.8 ESQUEMA COMPARACIONES PERMITIDAS

		SEGUNDO OPERANDO				
		GRUPO (QR)	ALFANU (AN)	N. ZONA (NZ)	N. EMPA (NP)	N. BINA (NB)
S U J E T O	GRUPO (Compues.) (QR)	NN	NN	NN	NN	NN
	ALFANUMERICO (AN)	NN	NN	NN (1)	*	*
	NUMERICO ZONA (NZ)	NN	NN (1)	CN	CN	CN
	NUMERICO EMPAQ. (NP)	NN	*	CN	CN	CN
	NUMERICO BINARIO (NB)	NN	*	CN	CN	CN



NN: Comparación No Numérica, se realiza de izquierda a derecha en función del código EBCDIC.

CN: Comparación Numérica.

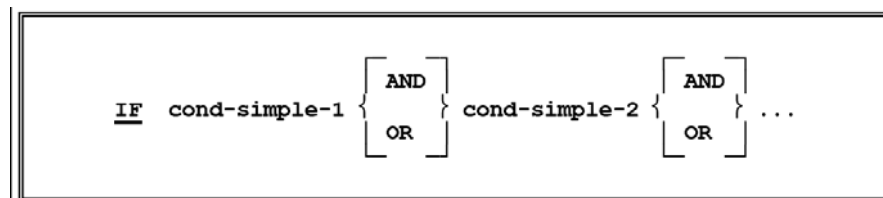
(1) Permitido si son valores numéricos enteros.

\*: Error de compilación.

## 2.9 CONDICIONES COMPUESTAS

Son series de dos o más CONDICIONES DE RELACIÓN conectadas lógicamente mediante los operadores lógicos **AND** y **OR**.

FORMATO SIMPLIFICADO:



**IF:** Verbo de la sentencia.

**Cond-simple:** Condiciones de relación de cualquier tipo.

**AND:** Operador lógico o funtor PRODUCTO.

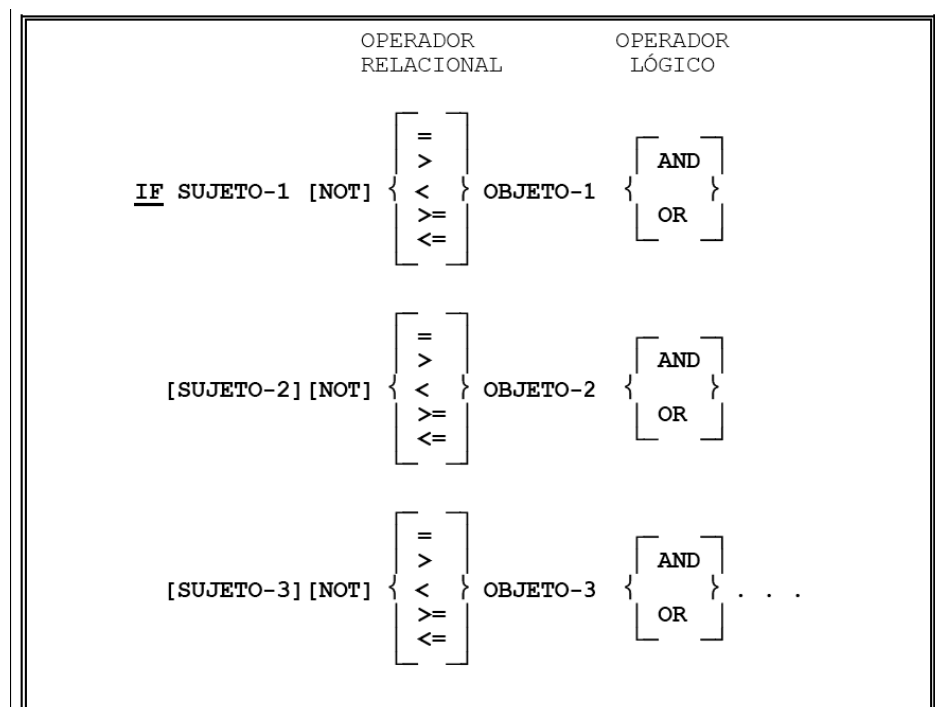
Todas las condiciones simples han de ser verdaderas para que la compues ta también lo sea. Sólo en tal caso se ejecuta la parte THEN del IF.

$$1 * 1 * 1 * 1 = 1 \quad 1 * 0 * 1 * 1 = 0$$

**OR:** Operador lógico o SUMA.

Basta con que una condición sea cierta para que la condición compuesta también lo sea. En tal caso se ejecutarán las declaraciones de la parte THEN.

$$0 + 0 + 1 + 0 = 1 \quad 0 + 0 + 0 + 0 = 0$$

**FORMATO AMPLIADO:****POSICION DE OBJETOS, OPERADORES Y OBJETOS DE LA COMPARACIÓN**

Si el **operador relacional** es EL MISMO en dos condiciones simples de una condición compuesta, **se puede omitir** la codificación de dicho operador relacional, ya que si no aparece un operador el compilador **asume** que es el mismo operador relacional que aparece en la condición simple anterior.

Si el **sujeto es el mismo** en varias relaciones simples de una compuesta, también se puede omitir su codificación, ya que, es lo que el compilador supone por defecto.

La codificación de SUJETO-1 y el primer operador relacional siempre es obligatorio reflejarlos, aunque los siguientes se asuman por defecto. El **objeto** de la comparación, también debe de escribirse siempre.

	4	6	7	8	12	16	20	24	28	32	36	40	44	48	52	56	60	68	72
140																			
150					IF	PRECIO	=	20		OR	PRECIO	=	80						
160					MOVE			A		TO	B								
180																			
190	*	****																	
200																			
205																			
210					IF	PRECIO	=	20		OR	80								
220					MOVE			A		TO	B								
220																			
230	*	****																	
230																			

## 2.10 ORDEN DE EVALUACIÓN DE CONDICIONES

- 1º. PARÉNTESIS (Si varios niveles, **de dentro hacia fuera**).
- 2º. Operadores de RELACIÓN.
- 3º. Operador lógico NOT.
- 4º. Operador lógico **AND**.
- 5º. Operador lógico **OR**.
- 6º. A IGUALDAD de operadores, **de izquierda a derecha**.

## 2.11 CODIFICACIÓN DE CONDICIONES Y LEYES DE MORGAN

El operador lógico **NOT**, se usa para negar una condición simple (de relación, de clase o de nombre de condición), o una condición compuesta.

El operador lógico **NOT** se puede codificar delante de la condición completa encerrada entre paréntesis.

Así pues:

**A OR B** es lo **contrario** de **NOT (A OR B)** y viceversa

**C AND D** es lo **contrario** de **NOT (C AND D)** y viceversa

**NOT A AND NOT B** es lo **contrario** de **NOT (NOT A AND NOT B)**

**NOT C OR D** es lo **contrario** de **NOT (NOT C OR D)**

Una condición compuesta puede **simplificarse** aplicando las ya conocidas LEYES DE MORGAN. Si se aplican las LEYES DE MORGAN, para simplificar las condiciones de la parte de la derecha del ejemplo de arriba.

Estas quedarían como sigue:

**NOT (A OR B)** es igual que **NOT A AND NOT B**

**NOT (C AND D)** es igual que **NOT C OR NOT D**

**NOT (NOT A AND NOT B)** es igual que **A OR B**

**NOT (NOT C OR D)** es igual que **C AND NOT D**

Si se prefiere, una condición puede **negarse directamente** aplicando las LEYES DE MORGAN. En tal caso:

**A OR B** es lo **contrario** de **NOT A AND NOT B**

**C AND D** es lo **contrario** de **NOT C OR NOT D**

**NOT A AND NOT B** es lo **contrario** de **A OR B**

**NOT C OR D** es lo **contrario** de **C AND NOT D**

NOTA: Es importante recordar, que un **IF** con una condición en afirmativo, produce exactamente el mismo efecto, si se niega dicha condición y a la vez, se intercambian las declaraciones de la parte **THEN** por las de la parte **ELSE** entre sí.

EJEMPLO:

	A	B
4 6 7 8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72	
005		
010		IF PRECIO NUMERIC AND PRECIO POSITIVE
020		PERFORM PRECIO-CORRECTO
030		ELSE
040		PERFORM PRECIO-INCORRECTO•
050		
060	★ *****	El "IF" que se codifica abajo haría exactamente lo mismo.
070		
080		IF PRECIO NOT NUMERIC OR PRECIO NOT POSITIVE
090		PERFORM PRECIO-INCORRECTO
100		ELSE
110		PERFORM PRECIO-CORRECTO•
120		

### 3.- VERBO PERFORM

#### **FUNCIÓN:**

INVOCAR a un **conjunto de instrucciones** localizadas a continuación de un nombre de párrafo, un grupo de párrafos o secciones y que se ejecuten tantas veces como creamos necesario.

Después de ejecutar la o las sentencias que contiene el párrafo llamado, se continúa la ejecución del programa, a partir de la instrucción codificada justamente detrás de la invocación.

PERFORM permite variar el orden secuencial de instrucciones, provocando un **salto con retorno**. Este verbo, es altamente flexible y se puede presentar con varios formatos.

#### **ELEMENTOS BÁSICOS DE LOS FORMATOS DE ESTA DECLARACIÓN:**

CÓDIGO OPERACIÓN: **PERFORM** (Ejecutar).

OPERANDO: **Nombre-de-PARRAFO**.

#### **3.1 FORMATO 1: PERFORM BÁSICO.**

##### **PERFORM nombre-párrafo**

Bifurcará a nombre-párrafo, se ejecutarán todas las DECLARACIONES que pertenecen a nombre-párrafo, y retornará a la DECLARACION siguiente a PERFORM.

	A	B
4 6 7	8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72
010	PROCEDURE DIVISION.	
120	LINEA-PRINCIPAL.	
130		
130	-	
130	-	
140	PERFORM 400-LEER.	
140	PERFORM 200-CALCULO.	
150	PERFORM 300-IMPRIMIR.	
160		
130	-	
160	-	
170	STOP RUN.	
180		
190	200-CALCULO.	
200	ADD SALARIO-EMPLEADO TO SALARIO-TOTAL.	
210	MOVE ZERO TO SALARIO-EMPLEADO.	
220	MOVE SALARIO-TOTAL TO L-TOTAL-W.	
230		
240	300-IMPRIMIR.	
250	MOVE L-TOTAL-W TO L-SALIDA.	
260	WRITE L-SALIDA AFTER POSITIONING SALTO.	
270	ADD 1 TO CONTA-LIN.	
310		
280	400-LEER.	
290	READ FICHERO AT END	
300	MOVE 'FIN' TO FIN-FICHERO.	
320	499-LEER-X.	
330	EXIT.	
130		
340	-	

### 3.2 FORMATO 2: PERFORM ... THRU

**PERFORM nombre-párrafo-1 THRU nombre-párrafo-2.**

Se bifurca a nombre-párrafo-1, se realizan todas las declaraciones que le siguen hasta nombre-párrafo-2, inclusive.

Entre nombre-párrafo-1 y nombre-párrafo-2, puede haber múltiples párrafos; los cuales también se ejecutarán.

Su uso es mayor o menor, en función de las normativas de los Centros de Cálculo. Veamos algunos ejemplos, basados en el ejemplo anterior.

	A	B
4 6 7	8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72
010	PROCEDURE DIVISION.	
130		—
130		—
020		—
030	PERFORM 400-LEER THRU 499-LEER-X.	
040 *	****	Este formato suele ser bastante usado.
050		—
050		—
050		—
060	PERFORM 200-CALCULO THRU 300-IMPRIMIR.	
070 *	****	Este produce el mismo resultado que los dos PERFORMS
080 *		del ejemplo anterior. No es conveniente hacer esto,
090 *		ya que se pierde claridad y puede inducir a errores.

### 3.3 FORMATO 3: PERFORM ... TIMES

<u>PERFORM</u>	nombre-párrafo	[ <u>THRU</u>	nombre-párrafo-2 ]
	{	número-entero	}
	{	nombre-campo	}
		TIMES.	

Ejecutar un nombre-párrafo o varios, tantas veces, como indique un número entero o el contenido de un campo.







	A	B
4 6 7 8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72	
100	PROCEDURE DIVISION.	
200		
500	PERFORM UNTIL FIN-FICHERO	
510	MOVE CAMPO-A TO CAMPO-B	
520	PERFORM CALCULAR-SUELDO	
530	PERFORM IMPRIMIR-DETALLE	
540	IF ANIO NOT = '1492'	
570	DISPLAY '** NO ES EL AÑO CORRECTO **'	
560	ELSE	
570	DISPLAY '** AÑO DESCUBRIMIENTO DE AMERICA **'	
570	END-IF	
580	END-PERFORM.	

### 3.6 SEUDO-INSTRUCCIÓN EXIT

#### FUNCIÓN:

Seudo-declaración que facilita la ENTRADA a un nombre-de-párrafo sin que se ejecute ninguna DECLARACIÓN.

Se suele codificar para evitar que se pueda solicitar la ejecución de un párrafo “vacío”.

#### CARACTERÍSTICAS:

Única declaración del nombre-de-párrafo.

#### FORMATO:

nombre-de-párrafo.

EXIT.

	A	B
4 6 7 8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72	
100	PROCEDURE DIVISION.	
200		
300	PERFORM RUTINA-CALCULAR THRU FIN-RUTINA-CALCULAR-X.	
400		
500	STOP RUN.	
600		
600	RUTINA-CÁLCULAR.	
810	MOVE CAMPO-A TO CAMPO-B.	
820	PERFORM CALCULAR-SUELDO.	
830	PERFORM IMPRIMIR-DETALLE.	
840	IF PRECIO-E NOT NUMERIC	
870	DISPLAY '** PRECIO ERRONEO **'.	
860	FIN-RUTINA-CALCULAR-X.	
870	EXIT.	
970		

#### 4. VERBO EVALUATE

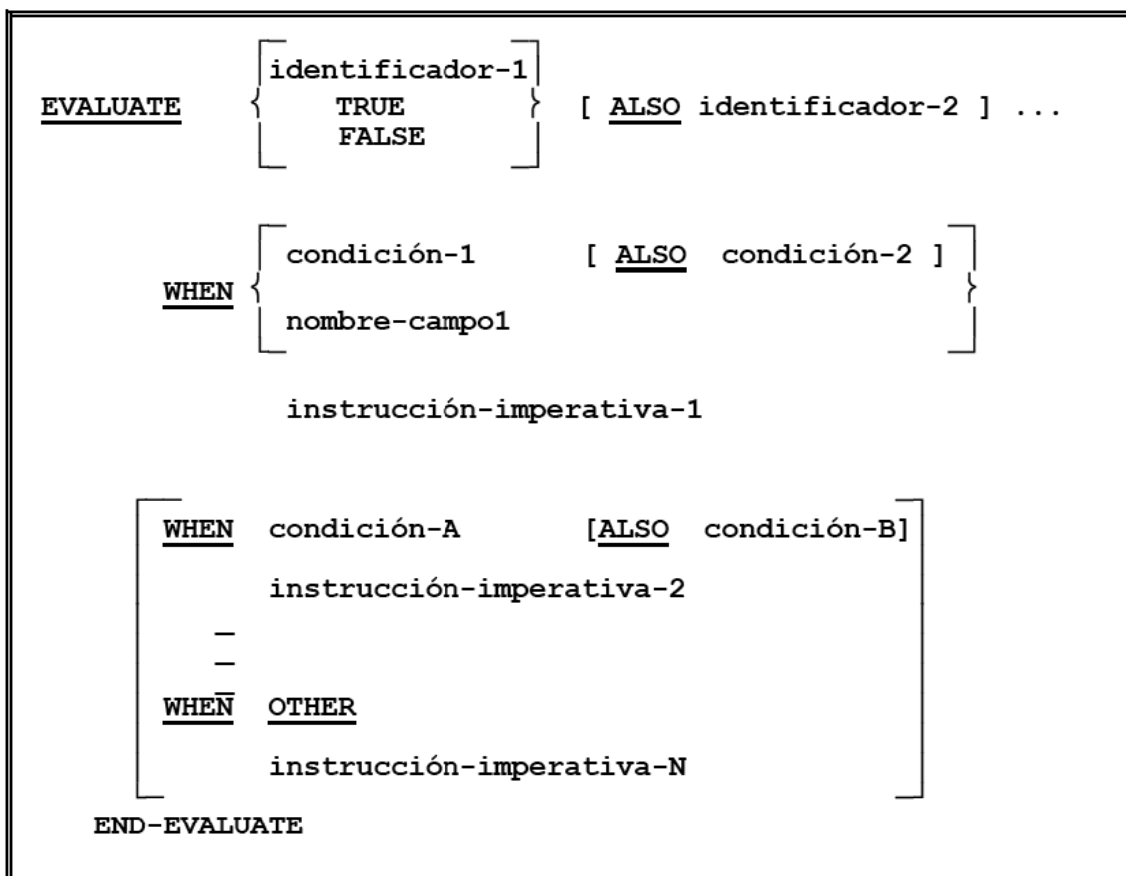
##### FUNCIÓN:

Permitir la implementación de **alternativas múltiples**.

Es la instrucción equivalente en COBOL a la **Estructura DO CASE** del pseudocódigo. Se suele usar, cuando se desea evitar la codificación de IF anidados.

Esta instrucción tiene una gran cantidad de opciones; para facilitar su comprensión, se opta por la representación del formato básico.

##### FORMATO:



**Identificador-1:** Es el campo o nombre de condición, cuyo valor se pretende evaluar.

**TRUE o FALSE:** Con TRUE se evalúa cuál es la primera condición que se cumple, y con FALSE la primera que es falsa.

**Condición-1:** Cualquier tipo de condición. En su lugar se puede poner UN LITERAL o un NOMBRE DE CAMPO.

**ALSO:** Se utilizará cuando sea preciso evaluar el valor de más de un campo.

**WHEN:** A esta cláusula, se asocia la condición y las sentencias que se ejecutarán, en caso de que esa condición se cumpla. Se pondrán tantos WHEN como se considere necesario.

**WHEN OTHER:** Opcional (en cualquier otro caso). Indica que hacer en caso de que ninguna condición anterior se hubiera cumplido.

**END-EVALUATE:** Determina donde termina el alcance de la instrucción EVALUATE. (El punto “.” haría lo mismo).

	A	B
4	6	7
8	12	16
20	24	28
32	36	40
44	48	52
56	60	64
72		

```

010 | WORKING-STORAGE SECTION.
020 | * =====
030 | 01 REG-W-E.
040 |     02 COD-ASIGNATURA-W          PIC 99.
050 |     02 CALIFICACION-W            PIC 99.
060 |         88 APROBADO    VALUE 5, 6.
070 |         88 NOTABLE     VALUE 7, 8.
080 |
090 | PROCEDURE DIVISION.
110 | * =====
210 |     EVALUATE CLAVE-ACCION
220 |         WHEN 'I'
230 |             PERFORM INCLUIR THRU INCLUIR-X
240 |         WHEN 'M'
250 |             PERFORM MODIFICAR THRU MODIFICAR-X
260 |         WHEN 'S'
270 |             PERFORM SUPRIMIR THRU SUPRIMIR-X
280 |         WHEN OTHER
290 |             PERFORM CLV-ERROR THRU CLV-ERROR-X
300 |     END-EVALUATE
310 | * - - - - -
320 |     EVALUATE TRUE
330 |         WHEN AÑO-E > AÑO-W
340 |             PERFORM RUTINA-AÑO-MAYOR
350 |         WHEN AÑO-E < AÑO-W
360 |             PERFORM RUTINA-AÑO-MENOR
370 |         WHEN AÑO-E = SPACES
380 |             PERFORM SOLICITAR-AÑO-E
390 |     END-EVALUATE
400 | * - - - - -
410 |     EVALUATE INDICE-E
420 |         WHEN INDICE-W PERFORM ACCEDER-A-TABLA
430 |     END-EVALUATE
440 | * - - - - -
450 |     EVALUATE COD-ASIGNATURA-W ALSO CALIFICACIÓN-W
460 |         WHEN 15 ALSO APROBADO
460 |             PERFORM PROCESAR-APROBADO
470 |         WHEN 1 THRU 14 ALSO NOTABLE
470 |             PERFORM PROCESAR-NOTABLE
480 |         WHEN 20 ALSO 5 THRU 10
480 |             PERFORM PROCESAR-ENTRE-5-Y-10
490 |         WHEN OTHER
490 |             PERFORM OTRA-SITUACION-DISTINTA
500 |     END-EVALUATE
510 | *
520 | * *****=> EL ALSO ES COMO UN AND.
530 |
540 |

```

## **5. SENTENCIA STOP RUN**

**FUNCIÓN:** Finalizar la ejecución del programa.

### **STOP RUN**

**STOP:** Verbo de la Sentencia.

**RUN:** El programa termina y retorna el control al sistema.

**ERRORES DE “EJECUCIÓN”  
DE PROGRAMAS “COBOL”  
EN  
ENTORNO “PC”**

***CÓDIGO EXPLICACIÓN DE LOS CÓDIGOS DE ERROR DE EJECUCIÓN  
de (Run-Time System Error Messages para Microsoft COBOL 3.0)***

***ERROR***

- 001** Problemas de buffer de memoria.
- 002** Se quiere leer un fichero cerrado.
- 003** Errores sucesivos. Se trata de un dispositivo en lugar de un programa.
- 004** Nombre de archivo inapropiado.
- 005** Especificación de dispositivo ilegal.
- 006** Se esta intentando escribir en un fichero abierto para leer.
- 007** Espacio insuficiente en el disco de destino.
- 008** Se esta intentando leer de un fichero abierto para escribir.
- 009** No más entradas en el directorio. El path suele estar mal codificado.
- 010** No se ha dado el nombre de archivo.
- 012** Se intenta abrir un archivo que ya está abierto.
- 013** Fichero no encontrado.
- 014** Demasiados ficheros abiertos. Ver FILES del config.sys.
- 015** Demasiados ficheros indexados abiertos.
- 016** Demasiados dispositivos abiertos.
- 017** Error en registro. Es probable que la longitud sea cero.
- 018** Se lee solo parte de un registro. Fichero abierto inadecuadamente.
- 019** Error al regrabar. Se accede o se abre incorrectamente un archivo.
- 020** Recurso o dispositivo ocupado por otro programa.
- 021** Se ha dado un nombre de directorio en lugar de un fichero.
- 022** Modo de acceso no permitido para el open realizado.
- 023** Modo de acceso no permitido para el close realizado.
- 024** Error de E/S en disco. Se quiere leer después de una escritura.
- 025** Se codifica a un terminal características que no son de un terminal.
- 026** Error de E/S en disco. Disco malo.
- 027** Dispositivo no disponible.
- 028** No hay espacio sobre el dispositivo.



- 029** Se intenta borrar un archivo que está abierto.
- 030** El fichero es de solo lectura. (Es un fichero del sistema).
- 031** No se tienen autorizaciones sobre el archivo.
- 032** Demasiados ficheros indexados.
- 033** Error de E/S físico.
- 034** El tipo de apertura es inapropiado.
- 035** Acceso a un archivo con permisos incorrectos.
- 036** El fichero no existe.
- 037** Acceso denegado a ese fichero.
- 038** El disco no es compatible.
- 039** El fichero no es compatible.
- 040** Inicialización incorrecta del lenguaje.
- 041** Fichero Indexado dañado.
- 042** Interface de E/S para escritura dañado.
- 043** Información incompleta para un archivo indexado.
- 047** Espacio insuficiente para un archivo indexado.
- 065** El fichero está bloqueado.
- 066** Clave de registro duplicada para un archivo indexado.
- 067** Fichero indexado no abierto.
- 068** El registro que se quiere leer está bloqueado.
- 069** Argumento no permitido para un archivo indexado.
- 070** Demasiados ficheros indexados abiertos.
- 071** Formato erróneo en un fichero indexado.
- 072** Final de fichero indexado.
- 073** Registro no encontrado en un fichero indexado.
- 074** Registro inválido para un archivo indexado.
- 075** Nombre de fichero demasiado largo.
- 076** No se puede crear un archivo bloqueado en un directorio indexado.
- 077** Error en módulo interno indexado.
- 078** La descripción de la clave es ilegal para el fichero indexado.
- 081** La clave ya existe en el fichero indexado.

- 100** Operación no permitida sobre un fichero.
- 101** Operación no permitida sobre un fichero indexado.
- 102** Fichero secuencial: longitud de registro incorrecta o no es secuencial.
- 104** No se ha dado el nombre de fichero en una operación sobre ficheros.
- 105** Error en una dirección de memoria.
- 106** Error interno o de E/S en un disco o en un fichero.
- 107** Operación no implementada sobre este sistema.
- 108** Fallo al inicializar la Data Division.
- 109** Se ha alterado la información interna del Sistema.
- 116** No es posible asignar memoria.
- 117** La secuencia de clasificación es errónea.
- 118** Incapaz de cargar un fichero objeto.
- 119** Incapaz de cargar un fichero objeto porque el punto de entrada es invalido.
- 120** Fichero objeto incorrecto.
- 121** Se quiere llamar a un subprograma que no es ejecutable.
- 122** Se intenta devolver el control de un call/perform por encima de la llamada.
- 123** Versiones de COBOL incompatibles.
- 129** Error se intenta acceder a un registro cero en un fichero relativo.
- 135** El fichero no existe.
- 138** No se puede abrir un fichero cerrado o bloqueado.
- 139** Longitud o clave del registro no correcta.
- 141** El fichero ya está abierto.
- 142** El fichero no se puede cerrar porque no esta abierto.
- 143** En un acceso secuencial para rewrite o delete no se ha leído previamente.
- 146** La posición del puntero es incorrecta. El read/start/invalid key falló.
- 147** El modo de apertura es inadecuado para una read o start.
- 148** Fichero abierto incorrectamente para realizar una operación write.
- 149** Fichero mal abierto para realizar una operación rewrite/delete.
- 150** El programa ha sido parado por el usuario.
- 151** Acceso random para un fichero secuencial.
- 152** Se quiere hacer un rewrite sobre un fichero que no se ha abierto como I/O.

- 153** El subíndice está fuera de rango.
- 154** Se ha superado el nivel de anidamiento de un perform.
- 155** Operación incorrecta.
- 156** Demasiados paréntesis en una sentencia compute.
- 157** La memoria es insuficiente porque el fichero objeto es muy grande.
- 158** Rewrite no permitido en un archivo con organización secuencial.
- 159** Fichero con organización LINE SEQUENTIAL incorrecto.
- 160** Error al cargar un módulo independiente.
- 161** Código intermedio no válido.
- 162** División por cero. Desbordamiento de memoria.
- 163** El valor de un campo numérico no es valido para su picture.
- 164** El subprograma que se está llamando no existe.
- 165** Versión de COBOL incompatible.
- 166** Proceso recursivo. Se está llamando a un módulo activo desde dentro de él.
- 167** Demasiados elementos a continuación de USING.
- 168** El tamaño de la pila ha sido sobrepasado.
- 169** Se intenta ejecutar una operación invalida.
- 170** Programa de utilidad del Sistema no encontrado: ADIS, LINK, IXSIO, ...
- 171** Operación japonesa no permitida en esta versión.
- 172** Final de dos perform con la misma dirección de retorno.
- 173** El fichero no existe en el directorio.
- 175** Se quiere ejecutar un programa con errores graves de compilación.
- 176** Referencia inadecuada de un segmento interno.
- 177** Intento de cancelar un programa activo.
- 178** Error al salvar la información generada por el programa.
- 179** Programa no encontrado. Error al ejecutar un "chain".
- 180** Falta la marca de fin de archivo en el fichero.
- 181** Error. Parámetro invalido.
- 182** Error. Se intenta leer de la pantalla o escribir en el teclado.
- 183** Se intenta abrir un archivo secuencial como I/O.
- 184** Error de E/S. Falta el módulo ADIS para realizar un ACCEPT o DISPLAY.

- 187** Path incorrecto. Ficheros no encontrados.
- 188** Nombre de archivo demasiado largo.
- 189** Error al cargar un módulo intermedio.
- 190** Demasiados argumentos en una CALL.
- 191** El tipo de terminal no está definido.
- 192** La configuración del terminal no tiene la operación requerida.
- 193** Se quiere ejecutar un fichero erróneo o que no existe.
- 194** Tamaño del fichero demasiado grande.
- 195** Delete/Write no precedido por un Read.
- 196** Número de registro demasiado grande (ficheros relativos/indexados).
- 197** El adaptador de video mal configurado.
- 198** Fallo al cargar un fichero.
- 199** Error indefinido del sistema.
- 200** Error lógico interno. El sistema se para.
- 201** Espacio insuficiente en el disco para paginar el fichero.
- 202** El código producido en algún preproceso o función interna es invalido.
- 203** Faltan parámetros en la CALL.
- 206** Se quiere leer datos que no existen de un fichero imagen.
- 207** Intento de acceder a una máquina que no está conectada.
- 208** Error en sistema multiusuario.
- 209** Error de comunicación en red local.
- 210** El fichero está bloqueado o está cerrado.
- 211** Programa ensamblado con una librería errónea.
- 212** Rutina en ensamblador con formato incorrecto.
- 213** Demasiados registros bloqueados.
- 214** Sentencia GO TO/ALTER mal empleadas.
- 215** Se intenta ANIMAR un programa con módulos de comunicaciones.
- 216** No se inicializar el dispositivo de comunicaciones referenciado.
- 217** Instrucción incompatible en esta versión de COBOL.
- 218** Fichero MULTIPLE REEL/UNIT mal creado.
- 219** El número de ficheros que pueden compartirse ha sido sobrepasado.

**220** Se pretende realizar simultáneamente más de un SORT/MERGE.

**221** Error en SORT/MERGE. (Ver status key).

**222** Error en SORT/MERGE. (Ver status key).

**223** Error en SORT/MERGE. (Ver status key).

**254** El programa ha sido cancelado desde teclado mientras se ANIMA el pgm.

**FIN**

## ***FILE EXPLICACIÓN DE LOS CÓDIGOS “FILE-STATUS KEY” DE FICHEROS STATUS CLASICOS EN “COBOL” Y ENTORNO “PC”***

### **ARCHIVOS SECUENCIALES:**

**‘00’** OPERACIÓN CORRECTA. La operación se ha realizado correctamente.

**‘10’** FIN ARCHIVO (AT END). Detectado el fin de fichero. No hay más registros.

**‘90’** OPERACIÓN INVALIDA. La acción que se desea realizar con un archivo, no es posible para el modo de apertura efectuado.

**‘91’** ARCHIVO NO ABIERTO. La acción que se desea hacer con el fichero, no es posible ya que dicho fichero esta cerrado.

**‘92’** ARCHIVO NO CERRADO. Se quiere abrir un archivo que ya está abierto.

**‘94’** APERTURA INVALIDA. Se intenta hacer un open de un archivo que no existe o no coincide con las características que se la han especificado.

### **ARCHIVOS INDEXADOS Y RELATIVOS:**

**‘00’** OPERACIÓN CORRECTA. La operación se ha realizado correctamente.

**‘02’** OPERACIÓN CORRECTA. PERO HAY REGISTROS CON CLAVE DUPLICADA. Si da al leer, indica que ya existe una clave con ese valor. Si da al escribir, indica que se ha grabado una clave con un valor idéntico a otro ya existente.

**‘10’** FIN ARCHIVO (AT END). Detectado el fin de fichero. No hay más registros.

**‘22’** ERROR VALOR DE CLAVE DUPLICADA. Se intenta grabar un registro con un valor de clave principal que ya existe en un archivo que no lo admite.

**‘23’** NO EXISTE EL REGISTRO QUE SE QUIERE LEER. (Opción Invalid Key).

**‘90’** OPERACIÓN INVALIDA. La acción que se desea realizar con un archivo, no es posible para el modo de apertura efectuado. O no se ha leído antes de hacer un DELETE o un REWRITE.

**‘91’** ARCHIVO NO ABIERTO. La acción que se desea hacer con el fichero, no es posible ya que dicho fichero esta cerrado.

**‘92’** ARCHIVO NO CERRADO. Se quiere abrir un archivo que ya está abierto.

**‘94’** APERTURA INVALIDA. Se intenta hacer un open de un archivo que no existe o no coincide con las características que se la han especificado.

**‘96’** NO SE HA DEFINIDO EL INDICADOR DE REGISTRO ACTUAL. Se está realizando un READ

y el resultado de la anterior READ o START no se ejecutó correctamente.

**‘98’** EL ÍNDICE ES INCORRECTO. El contenido del índice del fichero indexado no es válido para la instrucción de E/S que se quiere realizar.

**NOTA:** La cláusula **FILE-STATUS** nos facilita un medio adicional para conocer la respuesta dada por el sistema y los métodos de acceso a los datos sobre cada una de las instrucciones de E/S que se realizan contra un archivo (**OPEN, READ, WRITE, ...**).

El sistema introduce el código de terminación de cada petición en una variable declarada en la **working-storage section** cada vez que se ejecuta una instrucción de E/S. Dicho código indica al programa si la instrucción se ejecutó adecuadamente. (Es preferible usar las opciones **AT END, NOT AT END, INVALID KEY, ...**).

Cada vez que se compila un programa **COBOL** si tiene errores se genera un fichero con el mismo nombre del programa y con extensión **.LST** con un formato parecido a éste:

```

* Microsoft COBOL Version 3.00  L2.0 revision 053 26-May-98 15:59 Page  1
*
PROGRAMA=> NOMINAS.CBL
.....
52 77 CAM-AUX      PIC X(19) VALUE SPACES.
**14-E***                      ( 0)**
**   Period missing. Period assumed.
.....
54 7Z FIN-FICH     PIC XX  VALUE SPACES.
**209-S**                      ( 0)**
**   Level error not 01, 49, 66. Unknown statement
.....
73 01 TAB-PALABRA  VALUE HIHH-VALUES.
** 44-S*****                      ( 1)**
**   Literal expected
.....
74  02 ELEM-TAB    OCURS  500 TIMES.
**233-S*****                      ( 1)**
**   Unknown data description qualifier
.....
144  OPEN  INPUT   QUIJOTEP
** 11-S*****                      ( 1)**
**   Reserved word missing or incorrectly used
.....

```

### ANIMA.BAT NOMBRE-PROGRAMA (SIN EXTENSIÓN)

**F1 (Help):** Ayuda.

**F2 (View):** Ver pantalla de resultados en MS-DOS, para comprobar los datos enviados a pantalla desde el programa mientras se está ejecutando.

**F5 (Where):** El cursor vuelve a la línea activa de la procedure que se va a ejecutar. **ESC (Escape):** Salir del animador ó comando.

**Step :** Línea a línea de ejecución, según tecleamos **S**.

**Go:** Ejecuta línea a línea automáticamente, Para detener pulsar **Esc**.

**Zoom:** Ejecuta pgm hasta el final ó punto de parada.

**Reset:** Altera secuencia de ejecución: **Cursor-pitión:** Ejecuta desde donde está el cursor.

**Next:** Ejecuta la siguiente sin ejecutar la anterior.



**Start:** Comenzar de nuevo (se reinician otra vez todas las variables de la Working).

**Quit-perform:** Salir del perform sin ejecutarse.

**Break:** Puntos de parada.

**Set:** Poner punto de parada donde está el cursor (primera posición de la línea).

**Unset:** Quitar punto de parada donde está el cursor.

**Cancel-all:** Quitarlos todos.

**Examine:** Ver todos los puntos de parada.

**If:** Poner condición en el punto de parada.

**Do:** Incluir sentencia y se ejecuta.

**Query:** Visualizar contenido de campos.

**Q+C:** Visualiza contenido de la variable sobre la que esta situado el cursor.

**Q+R:** Visualiza último campo pedido.

**Q+Enter-name:** Introducir dato a visualizar.

**Locate:** Visualizar/localizar donde están definidos los campos en la Data División o los párrafos/rutinas en la Procedure División.

**L+C:** Se posiciona automáticamente en la variable sobre la esta posicionad el cursor.

**L+E:** Introducir (nom-fichero, nom-dato, nom-parrafo) que se quiere localizar.

**Do:** Ejecuta la sentencia que se teclea, incluyendo y ejecutándose inmediatamente.

**EJEMPLO DE PROGRAMAS COMPLETOS PARA PROGRAMAS LLAMADOS**

PROGRAMA LLAMANTE: SOLICITA EL LITERAL DEL MES EN CURSO Y LE PASO UN NÚMERO A TRAVÉS DEL CONTENIDO DE UN CAMPO

```
000100 IDENTIFICATION DIVISION.  
000200*=====
```

000300

```
000400 PROGRAM-ID. PEDIRMES.  
000500 AUTHOR. PEPITO PÉREZ.  
000600 DATE-COMPILED.  
000700 ENVIRONMENT DIVISION.  
000800*=====
```

000900

```
001000 CONFIGURATION SECTION.  
001100 SOURCE-COMPUTER. IBM-PC.  
001200 OBJECT-COMPUTER. IBM-PC.  
001300 SPECIAL-NAMES.  
001400 DECIMAL-POINT IS COMMA.  
001500  
001600  
001700 DATA DIVISION.  
001800*=====
```

001900

```
002100 WORKING-STORAGE SECTION.  
002200*_____
```

```
002300 77 LIT-MES PIC X(12) VALUE 'CUCUDRULU'.  
002400 01 FECHA-SYS.  
002500 02 AA PIC 99.  
002600 02 MM PIC 99.  
002700 02 DD PIC 99.  
004500 PROCEDURE DIVISION.  
004600*=====
```

```
004700 ACCEPT FECHA-SYS FROM SYSIN.  
004800 CALL 'DAMEMES' USING MM LIT-MES.  
004810 DISPLAY 'EL MES ACTUAL ES : ' LIT-MES.  
004820 STOP RUN.  
004900
```

RECIBE EL CONTENIDO DEL MES A TRAVES DE LOS CAMPOS DE ENLACE DE LA LINKAGE DEL OTRO PROGRAMA ÉSTE DEVUELVE CONTROL AL PROGRAMA LLAMADO PASÁNDOLE EL LITERAL DEL MES.

MIRAR LOS CAMPOS DE LA WORKING Y LA LINKAGE.

```
000100 IDENTIFICATION DIVISION.  
000200*=====
```

000400 PROGRAM-ID. DAMEMES.  
000500 AUTHOR. PEPITO PÉREZ.  
000600 DATE-COMPILED.  
002000 ENVIRONMENT DIVISION.  
002100\*=====

002300 CONFIGURATION SECTION.  
002400 SOURCE-COMPUTER. IBM-PC.  
002500 OBJECT-COMPUTER. IBM-PC.  
002600 SPECIAL-NAMES.  
002700 DECIMAL-POINT IS COMMA.  
003700 DATA DIVISION.  
003800\*=====

005900 WORKING-STORAGE SECTION.  
006000\*\_\_\_\_\_

006100 01 TABLA-MES.  
006200 02 FILLER PIC X(12) VALUE 'ENERO'.  
006300 02 FILLER PIC X(12) VALUE 'FEBRERO'.  
006400 02 FILLER PIC X(12) VALUE 'MARZO'.  
006500 02 FILLER PIC X(12) VALUE 'ABRIL'.  
006600 02 FILLER PIC X(12) VALUE 'MAYO'.  
006700 02 FILLER PIC X(12) VALUE 'JUNIO'.  
006710 02 FILLER PIC X(12) VALUE 'JULIO'.  
006720 02 FILLER PIC X(12) VALUE 'AGOSTO'.  
006730 02 FILLER PIC X(12) VALUE 'SEPTIEMBRE'.  
006740 02 FILLER PIC X(12) VALUE 'OCTUBRE'.  
006750 02 FILLER PIC X(12) VALUE 'NOVIEMBRE'.  
006760 02 FILLER PIC X(12) VALUE 'DICIEMBRE'.  
006770 01 FILLER REDEFINES TABLA-MES.

```
006780 02 LIT-MES PIC X(12) OCCURS 12.  
006790  
007500  
012100 LINKAGE SECTION.  
012120 77 MES-L PIC 99.  
012130 77 LIT-MES-L PIC X(12).  
012140  
012200 PROCEDURE DIVISION USING MES-L LIT-MES-L.  
012300*=====
```

012400 MOVE LIT-MES(MES-L) TO LIT-MES-L.

012500 GOBACK.

012600

# PROGRAMACIÓN CON TABLAS Ó ARRAYS

## 1.- CONCEPTOS BÁSICOS DE TABLAS.

Una de las estructuras de datos más utilizadas en la programación son los arrays o tablas. También se conoce como Vectores (a las tablas o arrays de una dimensión), como Matrices (a las tablas o arrays de dos dimensiones), como Poliedros (a las tablas o arrays de tres o más dimensiones).

**Tabla:** Estructura de datos constituida por un número fijo de elementos, todos ellos del mismo tipo y ubicados en direcciones de memoria físicamente contiguas.

### CARACTERÍSTICAS DE UNA TABLA

- ES UNA ESTRUCTURA DE DATOS DEFINIDA EN LA **MEMORIA PRINCIPAL** (**no** es un fichero).
- TIENE UN NOMBRE QUE IDENTIFICA A TODA LA TABLA. PARA ACCEDER A TODOS LOS ELEMENTOS DE LA TABLA A LA VEZ.
- TODOS Y CADA UNO DE LOS ELEMENTOS DE UNA TABLA TIENEN EL

### MISMO NOMBRE.

- TODOS Y CADA UNO DE LOS ELEMENTOS DE UNA TABLA SON DEL **MISMO TIPO** Y SU DEFINICIÓN ES IDENTICA.
- POR ESTO ÚLTIMO, (TODOS LOS ELEMENTOS SE LLAMAN IGUAL) ES IMPRESCINDIBLE UTILIZAR **UN ÍNDICE** O **APUNTADOR** PARA PODER REFERIRNOS E IDENTIFICAR A CUALQUIER ELEMENTO INDIVIDUALMENTE
- AL ACCEDER AL CONTENIDO DE UNA TABLA, EL ÍNDICE SIEMPRE DEBE TENER UN VALOR COMPRENDIDO ENTRE **1 Y EL NÚMERO MÁXIMO DE ELEMENTOS** QUE CONTIENE.

**Elemento:** Cada uno de los datos consecutivos que forman parte de la tabla.

**Nombre de la Tabla:** Es el identificador usado para referenciar la tabla y de forma global a todos los elementos que la forman.

**Tipo de una Tabla:** Marca el tipo de dato básico que es común a todos y cada uno de los elementos o componentes que forman dicha estructura (entero, real, carácter o lógico).

**Índices:** Valor numérico entero y positivo a través del cual podemos acceder directa e individualmente a los distintos elementos o componentes que la forman, marca la posición relativa de cada elemento dentro de la misma.

**Tamaño de la Tabla:** Longitud o número máximo de elementos que la forman.

**Acceso a los elementos o componentes de una Tabla:** Los elementos de una tabla reciben el mismo trato que cualquier otra variables simple, con un tipo de datos que coincide con el tipo de la tabla. Para acceder o referenciar a un elemento en particular es suficiente con indicar el nombre de la tabla seguido del índice correspondiente entre paréntesis.

**Dimensión de la Tabla:** Viene definida por el número de índices que necesitamos para acceder a cualquiera de los elementos que forman su estructura.

### **DEFINICIÓN DE UNA TABLA**

- Definir una tabla en MEMORIA PRINCIPAL implica establecer:
  1. NOMBRE SIMBÓLICO DE LA TABLA (para referencia global)
  2. NÚMERO DE ELEMENTOS QUE CONTIENE
  3. NOMBRE SIMBÓLICO DEL ELEMENTO (igual para todos).
  4. DEFINIR LA ESTRUCTURA DEL ELEMENTO (la misma para todos).
- LOS CAMPOS EN QUE SE SUBDIVIDE.
- LOS ATRIBUTOS Y LONGITUD DE CADA CAMPO.

### **BENEFICIOS DE LAS TABLAS**

- Es muy normal, en programación, copiar el contenido de ficheros en tablas cargadas en memoria Principal. Las ventajas más importantes que reporta cargar un fichero en una tabla para su posterior proceso son:

1. EL **ACCESO** A LOS DATOS DE UNA TABLA ES **RAPIDÍSIMO** EN COMPARACIÓN A LOS DATOS EN UN FICHERO.
  2. **SÓLO** TENEMOS **ABIERTO** EL FICHERO DURANTE LA CARGA.
- DE ESTA FORMA SE **PERMITE EL ACCESO** AL FICHERO OTROS USUARIOS y PROCESOS.

## 2.- CLASIFICACIÓN DE TABLAS

Según la estructura del elemento se clasifican en:

- **Simples:** el elemento no está subdividido en campos.
- **Compuestas:** el elemento está formado por varios campos.

Según su dimensión se clasifican en:

- **Unidimensionales:** Se necesita un índice para acceder al elemento.
- **Bidimensionales:** Se necesitan dos índices para acceder al elemento.
- **Tridimensionales:** Se necesitan tres índices para acceder al elemento.
- **Multidimensionales:** Se múltiples índices para acceder al elemento.

Según los valores de los elementos:

- **Ordenadas:** Sus elementos están ordenados.
- **Desordenadas:** Sus elementos están desordenados.

Según el número de elementos que tienen contenido:

- **Completa:** Todos los elementos tiene contenido.
- **Incompleta:** Algunos elementos están vacíos.

### 2.1.- TABLAS UNIDIMENSIONALES

También se les llama **vectores**. Los elementos se almacenan en posiciones contiguas adyacentes en la memoria principal de un ordenador.

Los elementos se almacenan en posiciones contiguas adyacentes en la memoria principal de un ordenador.

Ejemplos

TABLA-UNIDIMENSIONAL			
ELEM(1)	ELEM(2)	ELEM(3)	ELEM(4)

Otra forma típica de representarla gráficamente una tabla.

Tabla-Numeros contenidos) =>

Num(1)	Num(2)	Num(3)	Num(4)	Num(5)	Num(6)
8	14	26	38	110	212

**Acceso individual a los elementos de la tabla:**

Es suficiente su nombre seguido de **1 índice**.



**Nombre-Elemento(índice)**

Nombre-Tabla(índice) Válido en algunos lenguajes, en COBOL no.

**Acceso a los elementos de la Tabla-Numeros (un solo índice):**

- Accediendo al elemento **Num(1)** cogeríamos el valor que contiene **8**
- Accediendo al elemento **Num(2)** cogeríamos el valor que contiene **14**
- Accediendo al elemento **Num(3)** cogeríamos el valor que contiene **26**
- Accediendo al elemento **Num(4)** cogeríamos el valor que contiene **38**
- Accediendo al elemento **Num(5)** cogeríamos el valor que contiene **110**
- Accediendo al elemento **Num(6)** cogeríamos el valor que contiene **212**

**NOTA:** Algunos lenguajes consideran que el primer elemento se direcciona con valor de índice 0 (cero) y

otros con valor de índice 1. PARA NOSOTROS EL VALOR DEL PRIMER ÍNDICE SERÁ EL VALOR 1.

**2.2.- TABLAS UNIDIMENSIONALES COMPUESTAS**

También en estas tablas todos los elementos son iguales tanto en tipo como en nombre. Pero en una tabla compuesta, cada uno de los elementos de la tabla se subdivide en varios campos distintos, tanto en sus nombres como en sus definiciones, a los que se puede acceder por su nombre particular y un índice.

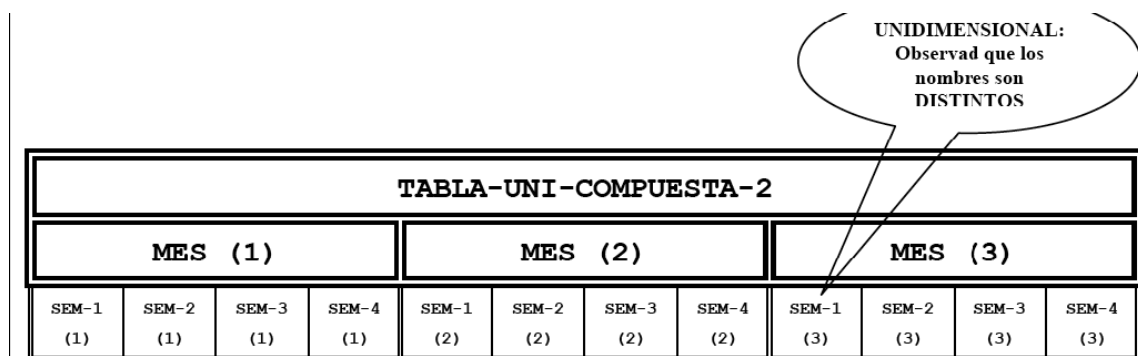
**Representación gráfica**

Ejemplo 1:

TABLA-UNI-COMPUESTA-1									
ELEMEN (1)		ELEMEN (2)		ELEMEN (3)		ELEMEN (4)		ELEMEN (5)	
COD (1)	CAN (1)	COD (2)	CAN (2)	COD (3)	CAN (3)	COD (4)	CAN (4)	COD (5)	CAN (5)

**Acceso a sus elementos de la Tabla-Uni-Compuesta-1 (un solo índice):**

- Accediendo a **COD(2)** cogeríamos el valor de esta variable.
- Accediendo a **CAN(5)** cogeríamos el valor de esta variable.
- Accediendo a **ELEMEN(4)** cogeríamos el valor de COD(4) y de CAN(4) simultáneamente.
- OJO: Los accesos a campos compuestos siempre son gestionados por el sistema como alfanumérico aunque cada variable individual fuese numérica.



UNIDIMENSIONAL:  
Observad que los nombres son DISTINTOS

TABLA-UNI-COMPUESTA-2											
MES (1)				MES (2)				MES (3)			
SEM-1 (1)	SEM-2 (1)	SEM-3 (1)	SEM-4 (1)	SEM-1 (2)	SEM-2 (2)	SEM-3 (2)	SEM-4 (2)	SEM-1 (3)	SEM-2 (3)	SEM-3 (3)	SEM-4 (3)

### Acceso a sus elementos de la Tabla-Uni-Compuesta-2 (un solo índice):

- Con **SEM-4(1)** accedemos al valor de la semana 4 del mes 1.
- Con **SEM-1(3)** accedemos al valor de la semana 1 del mes 3.
- Con **MES(2)** accedemos a la vez a las 4 semanas del mes 2.
- OJO: Los accesos a campos compuestos siempre son gestionados por el sistema como alfanumérico aunque cada variable individual fuese numérica.

## 2.3.- TABLAS BIDIMENSIONALES

Cada elemento o alguno de sus campos es, a su vez, una tabla.

También se les llama **matrices**.

Representación gráfica

Indices	Column. 1	Column. 2	Column. 3	Column. 4	Column. ...	Column. C
Fila 1	elem(1, 1)	elem(1, 2)	elem(1, 3)	elem(1, 4)	elem(1, ...)	elem(1, C)
Fila 2	elem(2, 1)	elem(2, 2)	elem(2, 3)	elem(2, 4)	elem(2, ...)	elem(2, C)
Fila ...	elem(..., 1)	elem(..., 2)	elem(..., 3)	elem(..., 4)	elem(..., ...)	elem(..., C)
Fila F	elem(F, 1)	elem(F, 2)	elem(F, 3)	elem(F, 4)	elem(F, ...)	elem(F, C)

Acceso individual a los elementos

Para acceder a uno de los elementos de esta tabla bidimensional es necesario utilizar **2 índices**: el 1º marca la fila, el 2º la columna.

**Nombre-tabla (índice-fila, índice-columna)**

Ejemplo: La tabla se llama NOTAS: 10 COLUMNAS > Alumnos del 1 al 10

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
F1	9	4	8	10	7	2	5	5	5	9
F2	8	10	7	5	9	4	8	10	0	1
F3	4	8	10	0	1	8	10	7	5	9
F4	7	5	9	4	8	10	0	5	5	5

4 FILAS > ASIGNATURAS (Matemáticas, Física, Historia y Filosofía)

### Acceso a los elementos de la tabla Notas (dos índices):

- Accediendo al elemento **Notas(4, 2)** cogeríamos el valor **10**

(Nota en **Filosofía** del **alumno 2**)

- Accediendo al elemento **Notas(3, 8)** cogeríamos el valor **7**

(Nota en **Historia** del **alumno 8**)

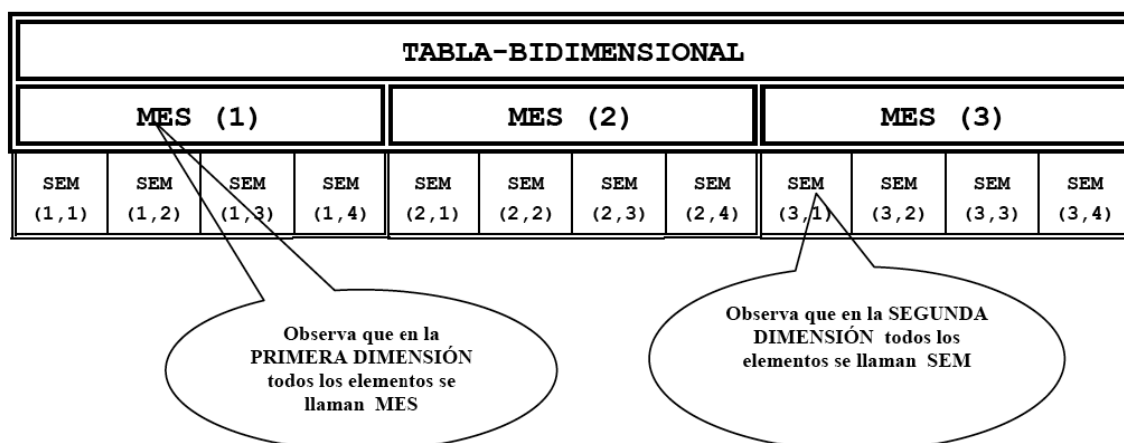
- Accediendo al elemento **Notas(2, 10)** cogeríamos el valor **1**

(Nota en **Física** del **alumno 10**)

NOTA: Los índices de fila y de columna nunca deben ser mayores de 4 y 10.

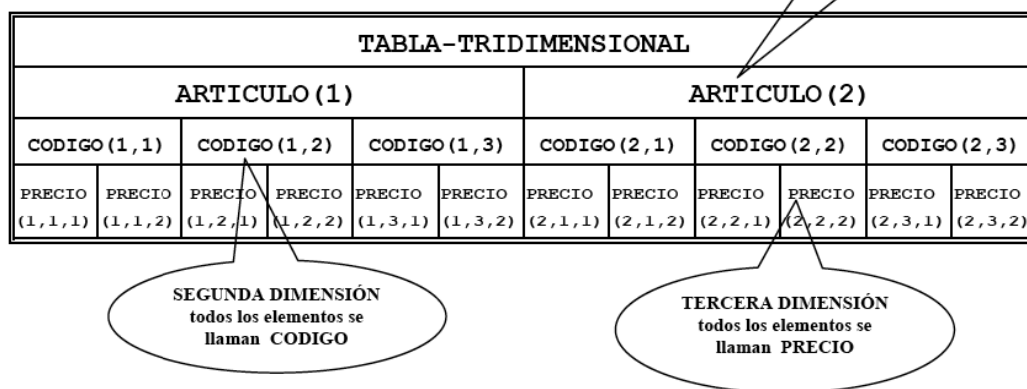
## 2.4.- EJEMPLO DE TABLAS BIDIMENSIONALES

Es una tabla con información de 3 meses y cada mes es una nueva tabla con información de los gastos acumulados de sus 4 semanas.



## 2.5.- EJEMPLO DE TABLAS TRIDIMENSIONALES:

Es una tabla con información sobre 2 artículos. Cada artículo se subdivide en otra tabla con 3 códigos y cada código tiene dos precios posibles.



## 3.- OPERACIONES TÍPICAS CON TABLAS Y FICHEROS

- Inicializar/ Preparar Tablas
- Cargar una Tabla desde un fichero
- Recorrido o acceso secuencial
- Búsqueda del contenido de un elemento
- \* En tablas desordenadas (búsqueda secuencial)
- \* En tablas ordenadas (búsqueda dicotómica)
- Ordenar una tabla

### 3.1.- EJEMPLO PRÁCTICO CON TABLAS.

Vamos a desarrollar a continuación los procesos más utilizados en programación con un ejemplo real con el siguiente organigrama:



El objetivo del programa es cargar parte del fichero de categorías en una tabla en memoria. El fichero sabemos que está desordenado. De los 25 campos que tiene el fichero solamente nos interesa cargar la categoría, la descripción y el salario asociado correspondiente el resto de campos del fichero no nos interesan en este programa. Sabemos que las categorías son 100 como máximo.

La misión del programa será cargar una tabla en memoria con la información seleccionada, después se ordenará la tabla para poder obtener un listado ordenado de esta información de categorías almacenada.

### 5.3.2.- DATOS DE PARTIDA SIMPLIFICADOS:

Registro de Entrada/Salida:

R-PERSO-E						
...	DES-E	...	SAL-E	...	CAT-E	...
	Campo5		Campo12		Campo24	

L-S | 132 posiciones alfanuméricas sin subdividir

Diseño del LISTADO de materiales solicitado:

	ASA	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2	3	4	5	6	7	8	9	40	
1																																										
2																																										
3																																										
4																																										
5																																										
6																																										
7																																										
8																																										
9																																										
10																																										
11																																										
12																																										
13																																										
14																																										
15																																										

## RESTO DE VARIABLES DE TRABAJO:

CAB-1	"LISTADO DE CATEGORIAS SALARIALES"					
CAB-2	"CATEGORIAS"		"DESCRIPCIÓN"		"SALARIO"	
CAB-3	"-----"		"-----"		"-----"	
L-D		CAT-D		DES-D		SAL-D
L-T	"TOTAL REGISTROS"				TOT-REG	

CONTA-REG	= 0
-----------	-----

FIN-FICHERO	= "NO"
-------------	--------

CONTA-REG	= 0	TOPE	= 0
-----------	-----	------	-----

IND	= 1	CAT-A-BUSCAR	"blancos"
-----	-----	--------------	-----------



ÚLTIMO  
ELEMENTO  
CON  
CONTENIDO

TABLA-CATEGORIAS											
ELEM-CATE-T (1)			ELEM-CATE-T (2)			...			ELEM-CATE-T (100)		
CAT-T (1)	DES-T (1)	SAL-T (1)	CAT-T (2)	DES-T (2)	SAL-T (2)	...	...	...	CAT-T (100)	DES-T (100)	SAL-T (100)

## EJEMPLO DE DEFINICIÓN DE ESTA TABLA EN LENGUAJE COBOL II:

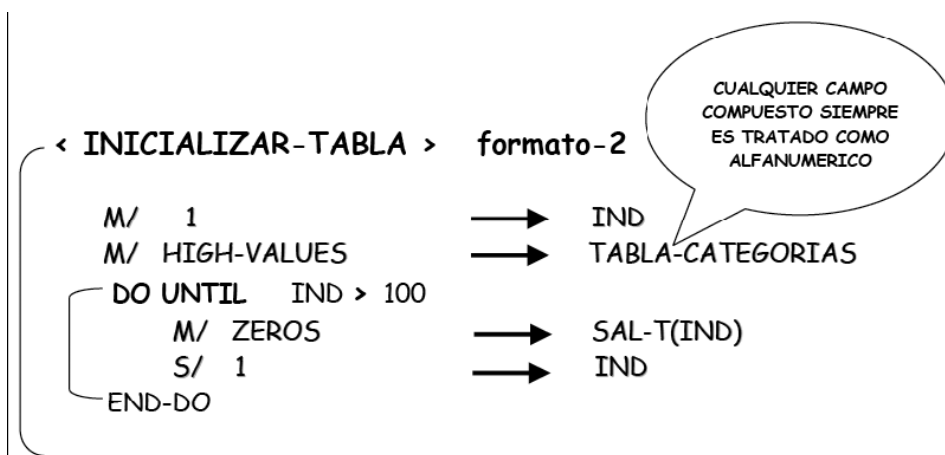
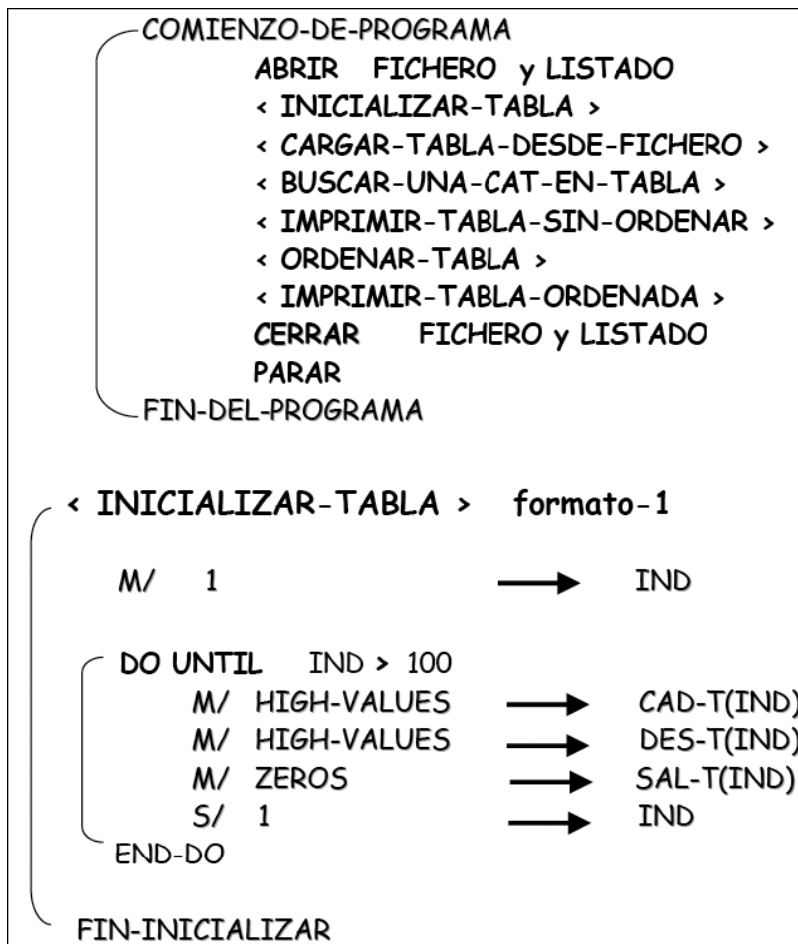
## 01 TABLA-CATEGORIAS.

## 02 ELEM-CATE-T OCCURS 100 TIMES.

03 CAT-T PIC X(06).

03 DES-T PIC X(25).

03 SAL-T PIC S9(10) COMP-3.



## &lt; CARGAR-TABLA-DESDE-FICHERO &gt;

M/ 1 → IND  
< LEER-FICHERO >

DO UNTIL FIN-FICHERO = "SI" OR IND > 100

M/ CAD-E → CAD-T(IND)

M/ DES-E → DES-T(IND)

M/ SAL-E → SAL-T(IND)

< LEER-FICHERO >

C/ IND = IND + 1

END-DO

IF FIN-FICHERO = "NO"

    MOSTRAR " ERROR LA TABLA ES PEQUEÑA"

    MOSTRAR " ERROR HAY REGISTROS SIN CARGAR"

    < CANCELAR >

END-IF

OPCIONAL: ÚLTIMO ELEMENTO CARGADO EN TABLA

C/ TOPE = IND - 1

FIN-CARGAR

## &lt; BUSCAR-UNA-CAT-EN-TABLA &gt;

ACEPTAR CAT-A-BUSCAR desde DISPOSITIVO-ENTRADA  
M/ 1 → IND

(( OR IND = 100 ))

DO UNTIL CAT-T(IND) = CAT-A-BUSCAR OR IND = TOPE

S/ 1 → IND

END-DO

IF CAT-T(IND) NOT = CAT-A-BUSCAR

    MOSTRAR " ERROR CATEGORIA NO ENCONTRADA"

    < CANCELAR-CATEGORIA-BUSCADA-NO-EXISTE >

ELSE

    MOSTRAR "OK. LA CATEGORIA ESTA EN: " IND

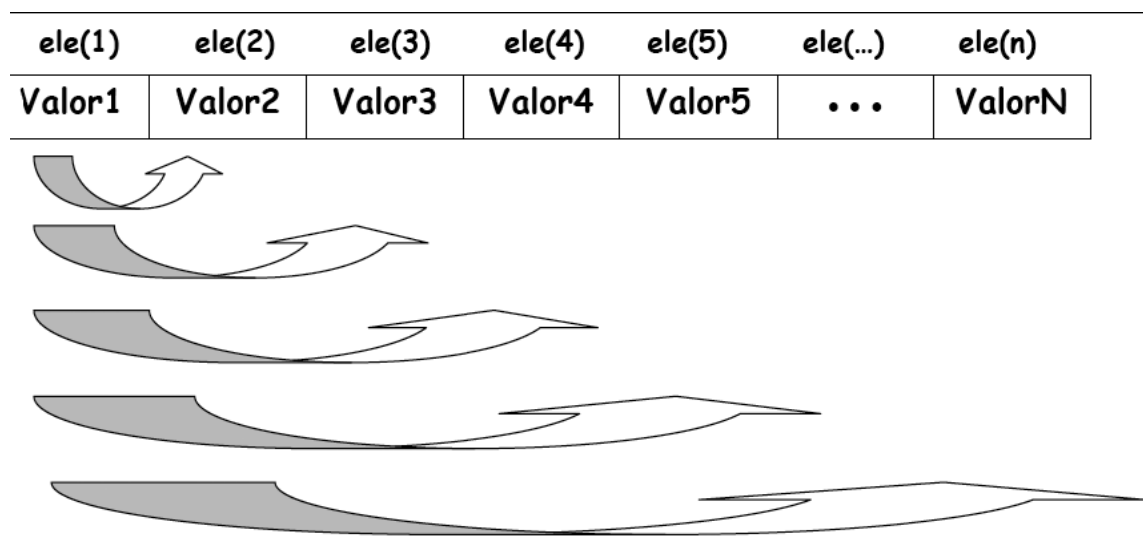
END-IF

DATOS INTRODUCIDOS  
DESDE EL TECLADO O  
DESDE OTROS SITIOS

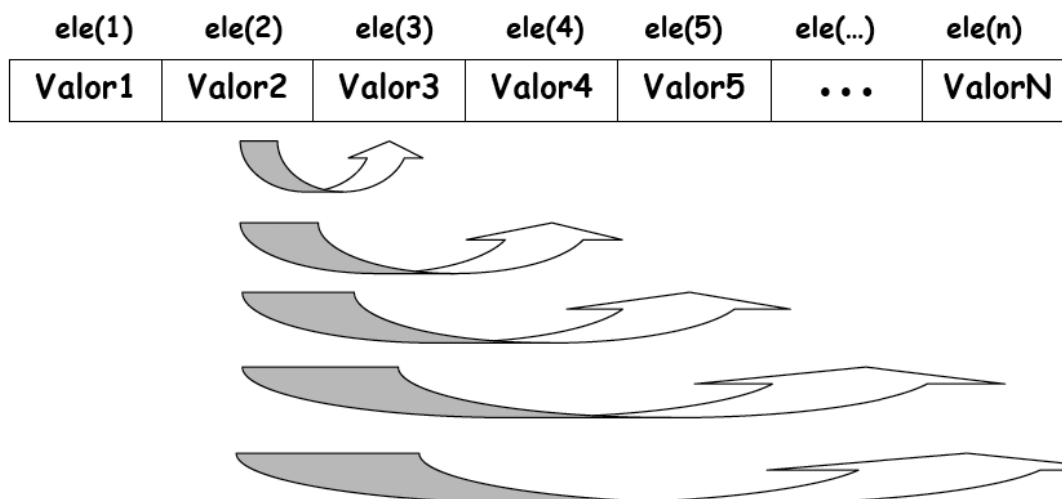


**ORDENAR LA TABLA: Método sencillo**

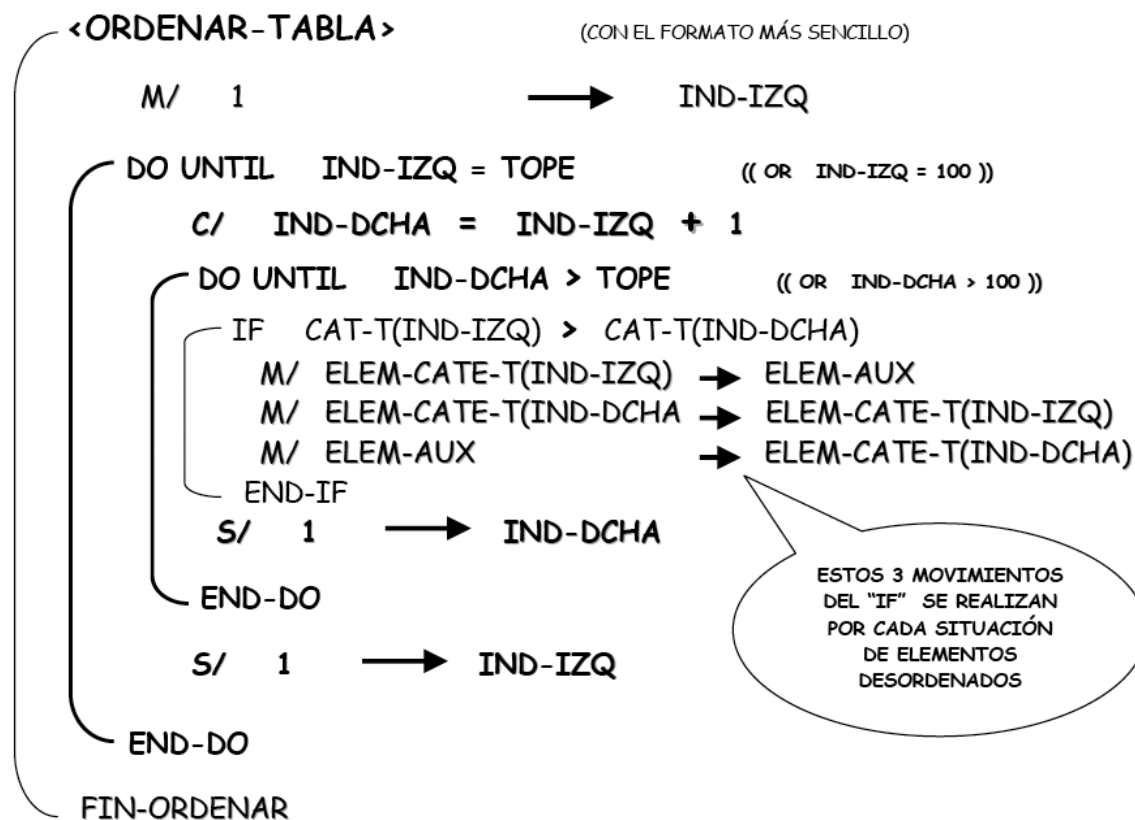
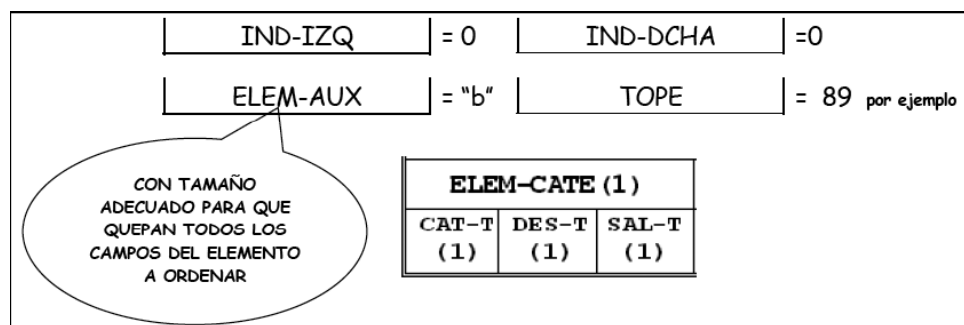
Con este método en la primera pasada se compara el primer elemento con todos los demás y, al final de ésta, el valor más pequeño queda en el primer elemento.



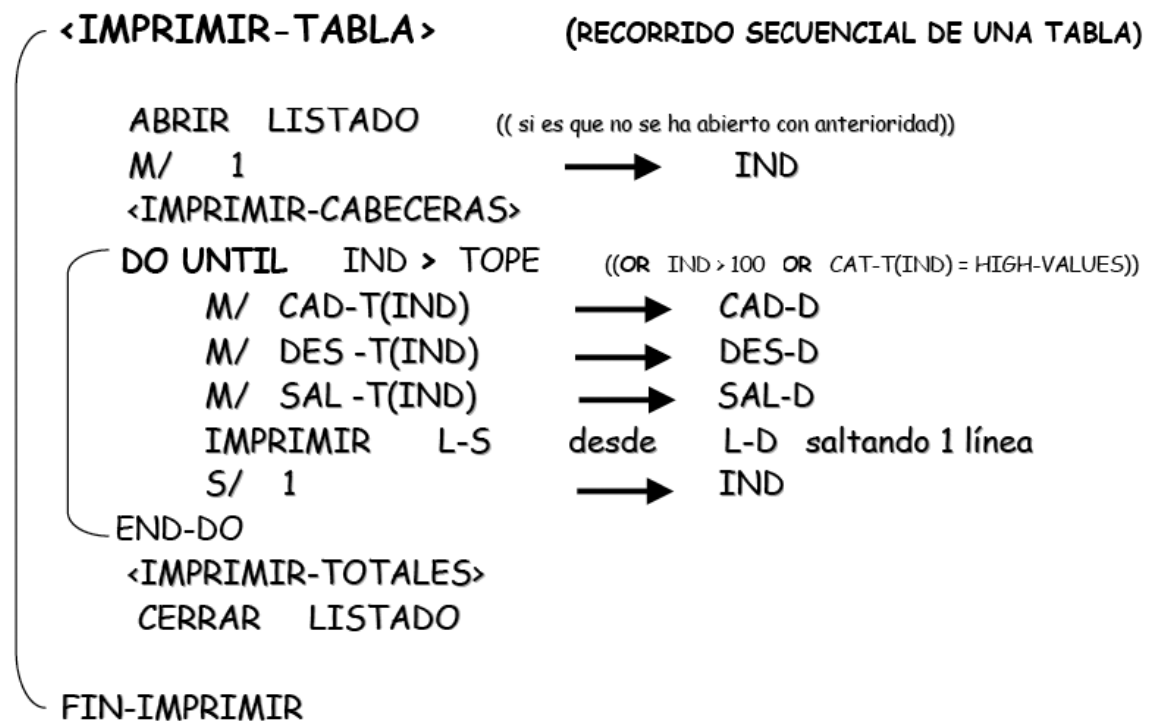
En la segunda pasada se compara el segundo elemento con todos los demás y al final de ésta el siguiente valor más pequeño queda ordenado en el segundo elemento y así sucesivamente.



## VARIABLES NECESARIAS PARA LA ORDENACIÓN DE LA TABLA



A continuación está el proceso que recorre secuencialmente todos y cada uno de los elementos de una tabla para cualquier cosa que se desee realizar con cada uno de ellos. Por ejemplo IMPRIMIR-TABLA con independencia de que esté ordenada o desordenada es indiferente.



# DEFINICIÓN DE FICHEROS

## 1 CARACTERISTICAS DE ENTORNO. ENVIRONMENT DIVISION.

### 1.1 FUNCIONES:

- Definir, asignar, direccionar y especificar peculiaridades de los ficheros que van a ser utilizados en el programa.

- Relacionar los “**nombres lógicos**” de los archivos, que van a ser utilizados por el programa.

Con los “**nombres reales**” de éstos en el dispositivo de almacenamiento.

- Especificar opcionalmente, las características y tipo del ordenador que se va a utilizar para “compilar” y “ejecutar” el programa.

	A	B
4 6 7	8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72
010		ENVIRONMENT DIVISION.
020	*	=====
025		
030		CONFIGURATION SECTION.
040		
050		SOURCE-COMPUTER. Comentario
060		
070		OBJECT-COMPUTER. Comentario.
080		
090		SPECIAL-NAMES.
100		DECIMAL-POINT IS COMMA.
110		
120		INPUT-OUTPUT SECTION.
020	*	=====
130		FILE-CONTROL.
140		
150		<u>SELECT</u> nombre-fichero-int <u>ASSIGN</u> TO nombre-externo
190		
200		SEQUENTIAL
210		<u>ORGANIZATION</u> IS { INDEXED }
220		RELATIVE
230		
240		SEQUENTIAL
250		<u>ACCESS</u> MODE IS { RANDOM }
260		DYNAMIC
370		
380		FILE <u>STATUS</u> IS nombre-campo.
390		
400		
410		

## 2 DESCRIPCIÓN DE SECCIONES

### 3.2.1 CONFIGURATION SECTION

Descripción de las características del ordenador, a nivel de COMENTARIO.

Ejemplo:

#### **SOURCE-COMPUTER. IBM-9000 ó IBM-3090**

Puede describir también el tamaño de memoria que ocupará el programa en palabras, caracteres o módulos.

Esta opción no es utilizada en ordenadores de gran capacidad. (Suele referirse a este tipo de ordenadores como entorno "HOST" y/o "MAIN-FRAME").

En esta Sección se pueden reasignar funciones.

Ejemplo:

#### **SPECIAL-NAMES.**

#### **DECIMAL-POINT IS COMMA.**

En cualquier campo numérico o numérico de edición, la coma hace funciones de punto y el punto de coma.

### 2.2 INPUT-OUTPUT SECTION

Permite enlazar los "**nombres simbólicos**" de los ficheros, (inventado por el programador), con los "**nombres reales**" de los mismos.

También, para dar la "**organización**", el "**tipo de acceso**" y **otras** características necesarias, para que todos los archivos puedan ser procesados adecuadamente.

#### **PÁRRAFO FILE-CONTROL**

**SELECT nombre-fichero-int ASSIGN TO nombre-externo.**

**SELECT nombre-fichero-int :**

Con la cláusula **SELECT** se definen cada uno de los ficheros del programa.

**ASSIGN TO :**

Con esta cláusula **nombre-fichero-int** se asocia a un **nombre-externo**, que será **el enlace**, que permitirá localizar el dispositivo de almacenamiento que soporta al Fichero Real.

**nombre-externo :**

**Nombre real** del fichero o **nombre de enlace** con él.

(Nombre real en entorno "PC" y nombre de enlace de 1 a 8 caracteres que aparecerá como nombre de la DD en el JCL, si es en el entorno "HOST").

### 3 CONCEPTO DE ORGANIZACIÓN

La organización hace referencia a la forma en que los registros lógicos están almacenados, o se van a almacenar, dentro del dispositivo de memoria auxiliar para ese fichero.

La elección de la organización de un fichero determina un mayor o menor uso de recursos del sistema, o una mayor o menor velocidad de procesamiento así como el tipo de acceso a la hora de procesar los registros.

La organización es decidida por el analista en función del tamaño del fichero y de la volatilidad y actividad de los registros lógicos.

Para soportar las distintas organizaciones, los sistemas operativos contienen un conjunto de rutinas especializadas llamadas métodos de acceso (secuencial, aleatorio o los dos).

#### 3.1 ORGANIZACIONES STANDARD DE FICHEROS (ANSI)

Son las organizaciones que suelen estar implementadas en todos los compiladores. (ANSI = AMERICAN NATIONAL STANDARD INSTITUTE).

#### 3.2 ORGANIZACIÓN SECUENCIAL (SEQUENTIAL)

Los registros lógicos son almacenados unos detrás de otros, por orden de llegada. Sólo se pueden procesar en forma secuencial.

Se pueden colocar en cualquier tipo de almacenamiento auxiliar.

El archivo sólo contiene registros lógicos (ninguna información secundaria).

#### 3.3 ORGANIZACIÓN INDEXADA (INDEXED)

Los registros son almacenados en orden ascendente de acuerdo con **el contenido de un CAMPO CLAVE predefinido**, que identifica a cada registro lógico.

Se pueden procesar tanto secuencialmente como aleatoriamente y sólo son posibles en dispositivos de acceso directo.

También se conoce a esta organización como SECUENCIAL-INDEXADA.

Los ficheros indexados están compuestos de un **área de datos** y un **área de índices** en la cual se guarda la clave de registro y la dirección de su posición en el área de datos. Esto permite el acceso directo a él.

### 3.4 ORGANIZACION RELATIVA (RELATIVE)

Se basa en la división del espacio físico de que disponga el fichero, en **celdas** destinadas a contener un solo registro. Estas celdas están numeradas por el **número de posición relativa** en el fichero, y podrán o no contener registro.

El usar archivos con organización relativa requiere que el programador desarrolle un **algoritmo de direccionamiento** que asocie la clave de los registros lógicos con el número de posición del registro en el fichero.

Es posible el proceso secuencial (en el orden del número de registro); también es posible el acceso directo a un registro, ya que, el sistema operativo es capaz de calcular la dirección en el disco a partir del número de registro.

Es imprescindible dispositivo de almacenamiento de acceso directo. Esta organización también recibe el nombre de “**aleatoria**”. El formato de esta declaración es:

#### ORGANIZATION IS tipo-de-organización

En la que se sustituye **tipo-de-organización** por:

- **SEQUENTIAL** Para ficheros de organización SECUENCIAL.
- **INDEXED** Para ficheros de organización INDEXADA.
- **RELATIVE** Para ficheros de organización RELATIVA.

\* Si no se especifica ninguna organización para el fichero, el sistema por defecto asume SEQUENTIAL.

### 4 MÉTODOS DE ACCESO

Los métodos de acceso a ficheros son el conjunto de técnicas que tienen por objeto **facilitar la búsqueda de información** dentro de la organización física de los ficheros. Esta función la realizan el Subsistema de Gestión de Ficheros integrado en el Sistema Operativo, que actúa como intermediario entre la “organización física” y la “organización lógica” (lo cual, es transparente al usuario).



Los principales métodos son: ACCESO SECUENCIAL, ACCESO DIRECTO y ACCESO DINÁMICO. El acceso Dinámico, se usa, para procesar un mismo fichero empleando los dos métodos de acceso anteriores a la vez.

#### 4.1 ACCESO SECUENCIAL (SEQUENTIAL)

Permite procesar registros secuencialmente, es decir, se leen o actualizan en el mismo orden en el que se encuentren. Para acceder a un determinado registro es imprescindible recorrer todos los anteriores hasta llegar a él.

Es el tipo de acceso que se asume por defecto.

#### 4.2 ACCESO DIRECTO (RANDOM)

Permite acceder directa y aleatoriamente a cualquier registro del fichero.

Es posible el acceso directo con ficheros Indexados (ISAM, y VSAM-KSDS) y también en ficheros Relativos (y VSAM-RRDS).

#### 4.3 SECUENCIAL Y DIRECTO A LA VEZ (DYNAMIC)

Este tipo de acceso en realidad es una mezcla de los dos anteriores.

Nos permite acceder **directamente** a un determinado registro lógico y después **secuencialmente**, todos los que estén grabados a continuación de él, hasta el final del fichero.

El formato, de la declaración es:

**ACCESS MODE IS modo-de-acceso**

Se sustituye **modo-de-acceso** por:

- **SEQUENTIAL** Para procesos secuenciales.
- **RANDOM** Para procesos de acceso directo.
- **DYNAMIC** Para procesos secuenciales y directos a la vez.

#### 4.4 CUADRO RESUMEN DE ORGANIZACIONES Y ACCESOS

TIPO DE FICHERO	TIPO DE ORGANIZACIÓN (ORGANIZATION)	MODO DE ACCESO (ACCESS)
<b>SECUENCIAL</b> (QSAM) (VSAM-ESDS)	SEQUENTIAL	SEQUENTIAL
<b>INDEXADO</b> (ISAM) (VSAM-KSDS)	INDEXED	SEQUENTIAL RANDOM DYNAMIC
<b>DIRECTO</b> (BDAM) (VSAM-RRDS)	RELATIVE	SEQUENTIAL RANDOM DYNAMIC

#### 5 DECLARACIÓN FILE STATUS

Esta cláusula **es opcional** en toda declaración de ficheros, indicará en qué campo debe depositar el Sistema Operativo el resultado de la ejecución de las instrucciones de entrada/salida mientras se está procesando un fichero. El valor resultante, indica si la instrucción se ejecutó correctamente o no.

##### FORMATO:

##### FILE STATUS IS nombre-campo

En la que **nombre-campo** :

- Deberá estar declarado en la WORKING-STORAGE SECTION como un campo alfanumérico de dos dígitos.

#### 6 CÓDIGOS DE FILE STATUS MÁS HABITUALES

##### 00 OPERACIÓN CORRECTA.

La instrucción se ha ejecutado satisfactoriamente.

##### 02 OPERACIÓN CORRECTA, PERO LA CLAVE TIENE DUPLICADOS.

Si se trata de una operación de lectura, indica que ya hay un valor de dicha clave. Si se trata de una grabación, indica que el registro que se acaba de grabar ha creado un valor duplicado de la clave.

#### **10 FIN DE ARCHIVO DE TECLADO (AT END).**

Se ha intentado leer un registro secuencialmente, cuando no existen registros en el archivo.

#### **22 VALOR DE CLAVE DUPLICADA.**

Se ha intentado grabar un registro que daría lugar a una clave duplicada, en un archivo donde no se admiten.

#### **23 NO SE HA ENCONTRADO EL REGISTRO.**

No existe el registro que se intenta leer.

#### **90 OPERACIÓN INVÁLIDA.**

Se ha intentado ejecutar una instrucción DELETE, READ, REWRITE, START o WRITE que está en desacuerdo con el modo de apertura o no se ha leído antes de una instrucción DELETE o REWRITE.

#### **91 ARCHIVO NO ABIERTO.**

El archivo no se encuentra abierto y no es posible ejecutar ninguna de las instrucciones especificadas.

#### **92 ARCHIVO NO CERRADO. ERROR LÓGICO**

Se ha intentado una instrucción OPEN para un archivo que ya está abierto o detectado el fin de fichero.

#### **94 OPEN INVÁLIDO.**

El archivo no se ha encontrado o no corresponde con las especificaciones.

#### **96 INDICADOR DE REGISTRO ACTUAL NO DEFINIDO.**

Se ha intentado una instrucción READ, cuando el resultado de la anterior instrucción READ o START no se ha podido ejecutar satisfactoriamente.

#### **98 INDICE INVÁLIDO.**

El contenido del índice de un archivo secuencial indexado no es válido para la operación de E/S que se pretende efectuar.

**NOTA:** Del 0 al 99 podrían variar, en función del fabricante del ordenador.

# PROGRAMACIÓN CON CURSORES

## 1. Introducción

Los cursores son necesarios para procesar sentencias **SELECT** en las que el resultado esté formado por varias filas.

Cuando el resultado de una **SELECT** contenga varias filas no se puede usar la sentencia **SELECT** embebida, puesto que los lenguajes de programación no están concebidos para el tratamiento en grupo de la información, sino para su tratamiento secuencial. En estos casos hay que emplear un mecanismo de programación, el **cursor**, que nos permite recuperar las filas y procesarlas secuencialmente en el programa.

También es posible actualizar datos por medio del posicionamiento del cursor, de forma que la sentencia de actualización afectará a la fila apuntada por el cursor.

El resultado de una sentencia **SELECT** es una tabla temporal, formada por todas las filas que satisfacen la condición de selección. Un cursor se comporta como un puntero, que se posiciona sobre las filas de esta tabla y las lee secuencialmente desde la primera a la última. La existencia de un orden determinado entre las filas, aunque se haya establecido arbitrariamente, implica que el cursor no se posicione dos veces en la misma fila durante su recorrido.

En cierta manera, un cursor se comporta como un fichero secuencial, formado por las filas de la tabla resultante de una sentencia **SELECT**, en un orden determinado. Por tanto se procesará de una manera similar:

hay que definirlo, abrirlo, procesar sus filas y cerrarlo. Estas operaciones se realizan con las siguientes sentencias SQL:

- **DECLARE** Define el cursor asociado a una sentencia **SELECT**.
- **OPEN** Abre el cursor, ejecutando la sentencia **SELECT** y construyendo una tabla temporal con un orden determinado.

- **FETCH** Avanza el cursor hasta la fila siguiente y deposita los valores de sus columnas dentro de las variables de trabajo del programa.
- **UPDATE** Actualiza columnas de la fila donde está posicionado el cursor.
- **DELETE** Borra la fila apuntada por el cursor.
- **CLOSE** Cierra el cursor, perdiendo el posicionamiento y la tabla temporal.

## 2. Definición del cursor

La definición de un cursor se realiza con la sentencia **DECLARE CURSOR**, cuyo formato es el siguiente:

```
EXEC SQL DECLARE n-cursor CURSOR FOR
      SELECT.....
      FROM n-tabla
      WHERE condición-de-selección
```

Donde “**n-cursor**” es el nombre que damos a ese cursor y que permite referirse a él con el resto de sentencias SQL relacionadas con los cursores, dentro del programa. Debe ser un nombre de 10 caracteres como máximo y que seguirá las normas del lenguaje de programación usado.

La sentencia **SELECT** que la definición del cursor lleva asociada tiene el mismo formato visto en capítulos anteriores (no lleva la cláusula INTO necesaria en el apartado 8.1). El resultado de esta sentencia es asociar un cursor a una sentencia **SELECT**, no ejecutarla.

En un programa se pueden definir tantos cursores como se deseen, incluso con idénticas sentencias **SELECT**, únicamente tienen que tener distinto nombre.

La sentencia **DECLARE CURSOR** debe preceder en el programa fuente al resto de sentencias sobre este cursor.

Su colocación en el programa difiere del lenguaje utilizado. Por ejemplo en Cobol puede situarse en cualquier punto de la **WORKING-STORAGE SECTION** o de la **LINKAGE SECTION** de la DATA DIVISION, también puede colocarse en la **PROCEDURE DIVISION**.

**Ejemplo:**

Declaramos el cursor C-EMPLEADO, para recuperar los datos de los empleados del departamento A00.

```
.....  
EXEC SQL DECLARE C-EMPLEADO CURSOR FOR  
      SELECT NUEMPL, NOMBRE, TLFN, SALARIO, SEXO  
      FROM TEMPL  
      WHERE DPTO = 'A00'  
      ORDER BY NUEMPL  
END-EXEC.
```

Nota: Como esta sentencia no ejecuta la SELECT, no es necesario preguntar por el código de retorno.

**3. Apertura del Cursor**

La apertura del cursor se realiza con la sentencia **OPEN**. Su formato es:

**EXEC SQL OPEN n-cursor**

Siendo “**n-cursor**”, el nombre del cursor previamente definido con una sentencia DECLARE CURSOR.

En el momento del OPEN es cuando el programa pasa al Gestor la petición de ejecución de la sentencia SELECT incluida en la sentencia DECLARE CURSOR. El Gestor ejecuta la SELECT y pone a disposición del programa la tabla temporal resultante de la ejecución.

Esto no implica que la tabla temporal exista físicamente. El Gestor puede construir la tabla completamente al ejecutar el OPEN y almacenarla en un fichero aparte mientras el cursor esté abierto, o bien ir construyéndola sobre la marcha según se le vayan pidiendo filas.

Además de construir “lógicamente” la tabla temporal, la apertura de un cursor provoca que éste se coloque apuntando a una posición anterior a la primera fila.

**Ejemplo:**

Abrimos el cursor definido anteriormente.

.....

```
EXEC SQL OPEN C-EMPLEADO END-EXEC.
```

```
    IF SQLCODE NOT = 0
```

```
    GO TO CANCELAR.
```

.....

**4. Lectura Secuencial de las Filas**

Para obtener una a una, secuencialmente, las filas de una esta tabla temporal la sentencia SQL que se usa es la FETCH. Su formato es:

```
EXEC SQL FETCH n-cursor
```

```
    INTO :n-campo1, :n-campo2,....
```

La cláusula **INTO** tiene aquí el mismo significado que se ha visto anteriormente.

Antes de ejecutar la sentencia **FETCH** el cursor debe haber sido abierto. Al ejecutarla el Gestor avanza la posición del cursor a la siguiente fila y extrae los contenidos de sus columnas depositándolos sobre las variables de trabajo de la cláusula INTO. En este momento el programa ya puede manipular los datos como desee.

Desde que un cursor se abre hasta que se cierra, la posición varia con cada una de las sucesivas sentencias SQL ejecutadas. En cualquier momento el cursor apuntará a una fila o al "hueco" que queda entre dos filas, si la fila a la que estuviera apuntando hubiera sido borrada usando el cursor. En cualquier caso al ejecutar una FETCH el cursor avanza hasta posicionarse en la siguiente fila y se queda ahí hasta que no haya otra sentencia SQL que le cambie de posición.

Cuando el cursor está posicionado sobre la última fila, el siguiente FETCH avanza intentando posicionarse sobre la siguiente fila, que no existe. Entonces el Gestor deja el cursor apuntando a un hueco posterior a la última fila y devuelve un código



de retorno **100** al SQLCODE. Si se siguen lanzando mas FETCH el resultado será el mismo que antes.

Cuando se detecta el SQLCODE 100 el programa deberá dejar de intentar leer nuevas filas y cerrar el cursor.

**Ejemplo:**

Leemos los empleados.

.....

LEER.

EXEC SQL FETCH C-EMPLEADO

    INTO :NUEMPL-W,

    :NOMBRE-W,

    :TLFN-W,

    :SALARIO-W,

    :SEXO-W

END-EXEC.

IF SQLCODE = +100

    TO FIN-TABLA

ELSE

    IF SQLCODE NOT = 0

        GO TO CANCELAR.

.....

GO TO LEER.

**5. Modificación de Datos**

Cuando un cursor está posicionado en una fila, después de ejecutar una sentencia FETCH, es posible modificar o borrar esta fila. Estas operaciones repercutirán directamente sobre la tabla original, para ello la sentencia SELECT incluida dentro de la definición del cursor debe cumplir ciertas restricciones, similares a las de las vistas actualizables.

Se dice que la tabla resultante de un cursor no es actualizable si la sentencia **SELECT** asociada tiene alguna de las siguientes características:

- Proviene de más de una tabla.
- Proviene de una vista no actualizable.
- Tiene agrupamientos (GROUP BY).
- Sus columnas son funciones predefinidas o expresiones aritméticas.
- Contiene subselects relacionadas.
- Usa la cláusula UNION.
- Usa la cláusula ORDER BY.

Si la tabla resultante es actualizable, se pueden utilizar las sentencias **UPDATE** y **DELETE** combinándolas con un cursor, produciéndose la actualización automática sobre la tabla original. Además siempre es posible actualizar una tabla directamente sin el uso de cursores, aun cuando ésta está siendo usada en el cursor.

**\* Borrado de filas con cursor.**

El formato de la sentencia **DELETE** cuando se borra la fila apuntada por el cursor, es similar al visto hasta ahora, sólo que en la cláusula WHERE no se especifica una condición de selección, si no el cursor que se usa para borrar.

**EXEC SQL DELETE FROM n-tabla**

**WHERE CURRENT OF n-cursor**

Esta sentencia borra la fila de la tabla que está apuntada, en ese momento por el cursor, que ahora apuntará al hueco que deja esa fila. Se borrará también la fila de la tabla original.

**\* Modificación de filas con cursor.**

La sentencia que se utiliza es la **UPDATE**, la única diferencia igual que antes está en la cláusula WHERE que especifica el cursor que se usa.

**EXEC SQL UPDATE n-tabla**

**SET.....**

**WHERE CURRENT OF n-cursor**

Esta sentencia actualiza la fila apuntada, en ese momento por el cursor, modificando las columnas indicadas en la cláusula SET.

Para usar este tipo de sentencia UPDATE es necesario, cuando definimos el cursor indicar que columnas de las filas resultantes pueden ser modificadas con el cursor. Para ello en la sentencia **DECLARE CURSOR** aparece una nueva cláusula, **FOR UPDATE OF**.

El formato completo de la sentencia DECLARE CURSOR queda entonces así:

```
EXEC SQL DECLARE n-cursor CURSOR FOR  
    SELECT.....  
    FROM n-tabla  
        WHERE condición-de-selección  
            FOR UPDATE OF n-columna1,  
                        n-columna2,..
```

Siendo “n-columna”, el nombre de cada una de las columnas que pueden ser modificadas usando el cursor.

#### **Ejemplos:**

- Vamos a borrar el empleado apuntado por el cursor después de ejecutar una sentencia FETCH.

.....

```
EXEC SQL FETCH C-EMPLEADO  
    INTO :NUEMPL-W,  
        :NOMBRE-W,  
        :TLFN-W,  
        :SALARIO-W,  
        :SEXO-W  
END-EXEC.  
    IF SQLCODE = +100  
        GO TO FIN-TABLA  
    ELSE  
        IF SQLCODE NOT = 0
```

```
                GO TO CANCELAR.  
                .....  
                .....  
                .....  
                .....  
EXEC SQL DELETE FROM TEMPL  
        WHERE CURRENT OF C-EMPLEADO  
END-EXEC.
```

```
        IF SQLCODE NOT = 0  
            GO TO CANCELAR.
```

```
        .....  
        .....  
        .....  
        .....
```

- Modificación del teléfono y el salario de un empleado apuntado por un cursor.

```
        .....  
EXEC SQL DECLARE C-MODIF CURSOR FOR  
        SELECT NUEMPL,  
                TLFN,  
                SALARIO  
        FROM TEMPL  
        WHERE SEXO = 'M'  
        FOR UPDATE OF TLFN, SALARIO  
END-EXEC.
```

```
.....  
.....  
.....
```

```
EXEC SQL FETCH C-EMPLEADO  
        INTO :NUEMPL-W,  
        :TLFN-W,
```

```
        :SALARIO-W
END-EXEC.
IF SQLCODE = +100
    GO TO FIN-TABLA
ELSE
    IF SQLCODE NOT = 0
        GO TO CANCELAR.
.....
.....
.....
        EXEC SQL UPDATE TEMPL
            SET TLFN = :TLFN-W,
            SALARIO = :SALARIO-W
            WHERE CURRENT OF C-MODIF
        END-EXEC.
        IF SQLCODE NOT = 0
            GO TO CANCELAR.
.....
.....
.....
```

## 6. Cierre del cursor

La sentencia que cierra el cursor es **CLOSE**, con el siguiente formato:

**EXEC SQL CLOSE n-cursor**

Al ejecutar esta sentencia se pierde el posicionamiento y la tabla resultado de la SELECT del cursor. Después de abrir un cursor, este puede cerrarse en cualquier momento, aunque lo normal es hacerlo al detectar el fin de la tabla con el SQLCODE igual a **100**.

En caso de que no se ejecute esta sentencia, al finalizar el programa el Gestor cerrará los cursores que queden abiertos.

**Ejemplo:**

Cerramos el cursor.

.....

EXEC SQL CLOSE C-EMPLEADO END-EXEC.

IF SQLCODE NOT = 0

GO TO CANCELAR.

.....

# PROCEDURE DIVISIÓ: CODIFICACIÓN DE INSTRUCCIONES

## 1. PROCEDURE DIVISION: CODIFICACIÓN DE INSTRUCCIONES

### 1.1. PROCEDURE DIVISION

#### FUNCIÓN

En esta division se **codifican las instrucciones** o sentencias necesarias para el desarrollo de la lógica del PROGRAMA.

Así pues, la procedure estará formada por todas las **instrucciones** que el ordenador deberá “ **ejecutar** “ con los datos de entrada definidos en las secciones anteriores, para obtener los resultados de salida apetecidos.

Las distintas instrucciones se pueden reagrupar, en función del tipo de tratamiento que realizan con los datos, en varios grupos.

Las Instrucciones codificadas en COBOL, a diferencia de otros lenguajes, sirven como AUTODOCUMENTACIÓN del programa ya que tienen la apariencia del lenguaje corriente (aunque en inglés). Esto, sumado a que es un lenguaje fácil de aprender y de mantener, explica por qué el 80% de las aplicaciones que se desarrollan en todo El Planeta, sean en COBOL.

### 1.2 TIPOS DE INSTRUCCIONES

- \* Instrucciones de Entrada/Salida.
- \* Instrucciones de Manipulación de Datos.
- \* Instrucciones para Cálculos Aritméticos.
- \* Instrucciones de Organización y Control.
- \* Instrucciones de Manipulación de Caracteres.

### 1.3 ORGANIZACIÓN DE LA PROCEDURE DIVISION

La Procedure está formada, de mayor a menor por:

- **SECCIONES**

- **PARRAFOS**

- **SENTENCIAS**

- **DECLARACIONES.**

Esta división al igual que las anteriores divisiones del programa está dividida en SECCIONES. Estas, a su vez, están formadas por PARRAFOS y cada uno de



éstos, contendrá una o más SENTENCIAS, las cuales se forman con una o más declaraciones.

Las INSTRUCCIONES del programa son las SENTENCIAS y las DECLARACIONES. A continuación se definen cada una de ellas.

### 1.4 DECLARACIONES

Se le da el nombre de DECLARACIÓN, a la unidad formada por un VERBO más sus operandos apropiados. Las declaraciones no terminan en punto.

			A	B														
4	6	7	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	72
010				OPEN			INPUT				FICHERO							
020																		
030				MOVE			CAMPOA				TO	CAMPOB						
040																		
050				WRITE			REGISTRO-NOMINA											
060																		

### 1.5 SENTENCIAS

Se denomina SENTENCIA, al conjunto formado por una o más DECLARACIONES acotadas por un PUNTO.

Ejemplos:

			A	B																
4	6	7	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	72		
010			PROCEDURE DIVISION.																	
010			MOVE	1				TO				X1.								
020																				
030			ADD	5				TO				X2.								
040																				
045			PERFORM	RUTINA-LEER				THRU				FIN-RUTINA-LEER.								
050																				
060																				
070			IF	CAMPO1				<	5											
080																				
090				MOVE				0				TO				X1				
100																				
110			ELSE																	
120																				
130				MOVE				6				TO				X1				
140				ADD				30				TO				CONTADOR.				
150																				
160																				

De todas las instrucciones del ejemplo, sólo hay una sentencia que engloba cuatro declaraciones, desde la línea 70 a la 140.

Las demás, también son sentencias, ya que acaban en punto, y están formadas por una sola declaración.

## 1.6 PÁRRAFOS

Se denomina PÁRRAFO, al **conjunto formado por**:

- \* UN NOMBRE de PÁRRAFO, que debe comenzar en el margen A, y que estará formado por la combinación de caracteres alfabéticos, numéricos y el guión. Un nombre de PÁRRAFO puede comenzar por número.

- \* UNA a más SENTENCIAS, situadas debajo de él.

Mediante el **nombre del párrafo** es posible referenciar a todas las sentencias que lo siguen en secuencia.

Un párrafo COMIENZA con su nombre y TERMINA delante de otro nombre de párrafo.

Los PÁRRAFOS se usan, (por regla general, en la programación estructurada) para crear, referenciar e identificar "LAS RUTINAS".

Ejemplo:

	A	B																	
4	6	7	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	72	
010			RUTINA-LEER.																
020			MOVE	SEQ-ACT										TO	SEQ-ANT.				
030			READ	FICHERO-HABITANTES															
040				AT END															
050								MOVE	HIGH-VALUES					TO	SEQ-ACT				
060								NOT AT	END										
070								MOVE	COD-PAIS-E					TO	PAIS-ACT				
080								MOVE	COD-REGION-E					TO	REG-ACT				
090								ADD	1					TO	CONT-LEIDOS.				
095																			
100			IF	SEQ-ACT				>	SEQ-ANT										
110				PERFORM				REGISTRO-ORDENADO											
120			ELSE																
130				PERFORM				FICHERO-DESORDENADO											
140				PERFORM				CANCELAR-PROCESO.											
150			FIN-RUTINA-LEER.																
160			EXIT.																

## 1.7 SECCIONES

Una sección es un nombre de párrafo seguido de la palabra reservada SECTION.

Una sección engloba a uno o más párrafos. Cuando se solicita la ejecución de una sección, el ordenador

intenta ejecutar todos los párrafos que estén debajo de ella, hasta que encuentra otro nombre de sección.

Las secciones son opcionales (excepto en la opción SORT) y el que se usen en mayor o menor grado, suele

depender de las normativas de los distintos centros de procesos de datos (CPD's).

Ejemplo:

	A	B
4 6 7 8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72	
010	PROCESAR-MAESTRO SECTION. <===	
020 *	Cuando invoca el párrafo PROCESAR-MAESTRO se ejecutarán todos	
021 *	los párrafos que están debajo de él, hasta otra sección.	
022		
023	APERTURA-FICHEROS.	
030	OPEN INPUT FIMAESTRO FIMOVIM	
040	OUTPUT FILISTADO.	
050	PROCESAR-REGISTROS.	
060	PERFORM LEER-MAESTRO.	
070	PERFORM LEER-MOVIMIENTOS.	
080	PERFORM ACTUALIZAR-FICHERO	
090	UNTIL MAESTRO = 'EOF' AND MOVIM = 'EOF'.	
100	.	
110	.	
120	.	
130	GO TO FIN-PROCESAR-MAESTRO.	
140	.	
150	.	
160	.	
170	LEER-MAESTRO.	
180	READ FIMAESTRO AT END ...	
190	.	
190	.	
190	.	
250	FIN-PROCESAR-MAESTRO.	
260	EXIT.	
200		
210	ROUTINAS-ORDENACION-MOVIMIENTOS SECTION. <===	
220	READ FIMOVIM AT END ...	
230	.	
240	.	
260	.	

## 2. INSTRUCCIONES DE ENTRADA/SALIDA

### **FUNCIÓN:**

Poner en comunicación al PROGRAMA con los DISPOSITIVOS EXTERNOS donde están o estarán almacenados los datos que se desean procesar.

### **TIPOS DE INSTRUCCIONES DE ENTRADA/SALIDA**

Las instrucciones de entrada salida son las siguientes:

#### **MISION VERBO**

« **ABRIR FICHEROS > OPEN**

« **CERRAR FICHEROS > CLOSE**

« **LEER FICHEROS > READ**

« **ESCRIBIR REGISTROS > WRITE**

« **ACEPTAR DATOS > ACCEPT**

« **VISUALIZAR DATOS > DISPLAY**

Ahora sólo veremos los verbos de Entrada de datos desde el monitor al programa (Accept) y de salida desde el programa al monitor (Display) para nuestros primeros programas de Cobol.

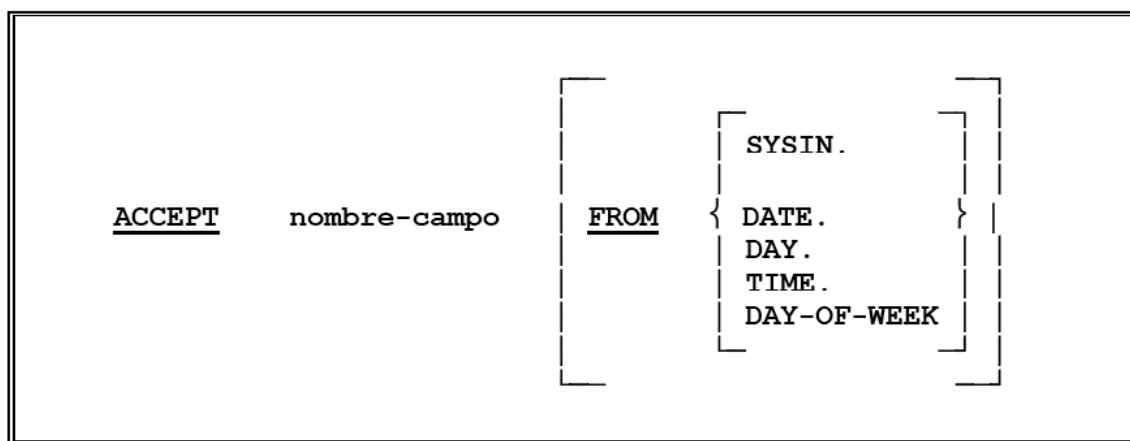
El resto de mandatos es para trabajar con Ficheros de entrada / salida que los veremos en otros módulos.

## 2.1 VERBO ACCEPT

### FUNCIÓN:

Facilitar la recuperación de datos **a través de los propios Recursos del Sistema.**

### FORMATO:



**ACCEPT:** Verbo de la Sentencia.

**nombre-campo:** Campo definido en la WORKING-STORAGE SECTION para recibir la información, desde el dispositivo por defecto (que de momento será la pantalla del ordenador) o a través de alguna de las opciones siguientes.

### OPCIÓN FROM SYSIN

En entorno "PC" el cursor se sitúa parpadeando en la pantalla del ordenador, el usuario teclea caracteres, y cuando pulsa la tecla ENTER, el sistema coloca los caracteres tecleados en el campo especificado a la derecha del verbo Accept

**Al terminar la operación el campo NOMBRE contiene el literal 'TOMAS '.**

Ejemplo:

```
WORKING-STORAGE SECTION.  
01  NOMBRE PIC X(10)  VALUE SPACES.  
  
PROCEDURE DIVISION.  
  ACCEPT TARJETA [FROM SYSIN].
```

Al terminar la operación el campo **NOMBRE** contiene el literal 'TOMAS '.

#### **OPCIÓN FROM DATE**

Se recupera la Fecha del día con este Formato **AAMMDD**.

Nombre-campo en este caso será un NUMERICO DISPLAY = **PIC 9(6)**.

#### **OPCIÓN FROM DAY**

Se recupera la fecha del día con este formato **AADDD**.

Nombre-campo en este caso será un NUMERICO DISPLAY = **PIC 9(5)**.

#### **OPCIÓN FROM TIME**

Se recupera la hora con este formato **HHMMSSDD**.

Nombre-campo en este caso será un NUMÉRICO DISPLAY = **PIC 9(8)**.

#### **OPCIÓN FROM DAY-OF-WEEK**

Se recupera un dígito que corresponde al día de la semana (1=LUNES, 2=MARTES, 3=MIERCOLES ... ).

Nombre-campo en este caso será un NUMERICO DISPLAY = **PIC 9**.

## 2.2 VERBO DISPLAY

### FUNCIÓN:

Facilitar la escritura de datos a través de dispositivos del Sistema. Que en el PC será siempre el monitor, es decir, la pantalla del ordenador.

### FORMATO DE LA SENTENCIA:

```

DISPLAY { [ literal-1...Literal-2 ]
          [ n-datos-1...n-datos-2 ] .

```

**DISPLAY:** Verbo de la sentencia.

**Literal y/o dato:** Literal o dato que se desea escribir. El USAGE del campo que soporta el dato puede ser de cualquier tipo. Si el USAGE es COMP-3 o COMP, se convierte previamente a DISPLAY. De cualquier forma NO hay EDICIÓN, por lo cual es conveniente **editar** los campos numéricos del DISPLAY.

Ejemplo:

```

01  DISPLAYAR.
02  CAM1    PIC  X(8)      VALUE  'COMIENZO' .
02  CAM2    PIC  X(3)      VALUE  SPACES .
02  CAM3    PIC  X(8)      VALUE  'NUMERICO' .
02  CAM4    PIC  S9(7)     VALUE  -1234567 .
.....

```

DISPLAY          DISPLAYAR.

RESULTADO:        COMIENZO    NUMERICO123456P

Pueden ser varios los campos o literales a displayar.



Ejemplo:

DISPLAY 'COMIENZO' CAMPO2 CAM3 CAM4.

RESULTADO: COMIENZO NUMERICO123456P

Se consigue el mismo resultado que en el ejemplo anterior.

### OTROS EJEMPLOS:

Ejemplo:

01 DISPLAYAR.

02 CAM1 PIC X(8) VALUE 'COMIENZO'.

02 CAM2 PIC X(3) VALUE SPACES.

02 CAM3 PIC X(8) VALUE 'NUMERICO'.

02 CAM4 PIC S9(7) VALUE -1234567.

.....

DISPLAY DISPLAYAR.

RESULTADO: COMIENZO NUMERICO123456P

Nos tenemos que dar cuenta que cuando un programa se ejecuta, el usuario que está delante de la pantalla del ordenador lo único que ve es lo que el programa muestra de la memoria del programa a través del comando display. Si sólo saco números, sin información textual, es difícil que la persona sepa de qué le están hablando. Por tanto siempre que mostremos información al exterior lo haremos:

DISPLAY 'EL NUMERO DE HIJOS ES : ' NUM-HIJOS.

DISPLAY 'EL NOMBRE ES : ' NOMBRE.

De la misma manera cuando solicitamos información a través de la pantalla, dijimos que el cursor se sitúa parpadeando para que introduzcamos información.

Pero no sabemos lo que nos pide. Por eso la manera es:

DISPLAY 'INTRODUCE EL NOMBRE : '

ACCEPT NOMBRE.

1

INTRODUCE NOMBRE :

2

INTRODUCE NOMBRE : TOMAS

Tras dar al ENTER el literal tecleado en este caso TOMAS viaja al campo NOMBRE

# PRÁCTICAS COBOL EN PC

EN ESTE APARTADO VAMOS A AMPLIAR NUESTROS CONOCIMIENTOS DE COBOL II Así como con los editores y compiladores.

VAMOS A SELECCIONAR 4 PROGRAMAS QUE REALIZAMOS EN EL COLABORAR DEL MÓDULO COBOL II.

**NOTA:** EL ALUMNO DEBE CREAR CON UN EDITOR COBOL ESTOS 4 PROGRAMAS y COMPILAR Y EJECUTAR HASTA QUE FUNCIONEN ADECUADAMENTE

### PROGRAMA 1

Leer 10 números e imprimir el mayor de ellos.

Aquí, el programa sólo guarda el número mayor. Los tecleados se van perdiendo. CON REPETITIVAS).

Inicio

DATOS

NumeroMayor	entero
NumeroTecleado	entero
Contador	entero

PSEUDOCODIGO

NumeroMayor = 0            ' no imprescindible  
Imprimir "Introduce DIEZ números y te digo cual es el mayor"

Imprimir "Introduce el primer número"  
Leer NumeroMayor  
Contador = 1

DO WHILE Contador < 10            ( UNTIL Contador = 10 )  
    Imprimir "Introduce otro número"  
    Leer NumeroTecleado

    IF NumeroTecleado > NumeroMayor then  
        NumeroMayor = NumeroTecleado

    END-IF

    Contador = contador + 1

END-DO

Imprimir "EL MAYOR DE LOS 10 ES: " NumeroMayor

Fin

## PROGRAMA 2

LO MISMO QUE ANTES PERO CON RUTINAS Y NIVELES 88 (BOOLEANOS)

Inicio Programa Principal

DATOS

figura,	alfanum	Apotema,	entero
Base,	entero	Lado,	entero
Altura,	entero	Area,	entero

PSEUDOCODIGO

Imprimir "Introduzca un nombre de figura, en minúsculas y sin acentos"

Leer figura

SELECT CASE figura

CASE "triangulo" "TRIANGULO" "T" "t"

<Hallar-Area-Triangulo>

CASE "cuadrado" "CUADRADO" "C" "c"

<Hallar-Area-Cuadrado>

CASE "pentagono" "PENTAGONO" "P" "p"

<Hallar-Area-Pentagono>

CASE ELSE

Imprimir "Figura no prevista"

END SELECT

Fin Programa Principal

RUTINA o subprograma: <Hallar-Area-Triangulo>

PSEUDOCODIGO

Imprimir "Introduzca la base del triángulo"

Leer Base

Imprimir "Introduzca la altura del triángulo"

Leer Altura

$Area = Base * Altura / 2$

Imprimir "El área del triángulo es: " Area

Fin subprograma

RUTINA o subprograma: <Hallar-Area-Cuadrado>

PSEUDOCODIGO

Imprimir "Introduzca el lado del  
cuadrado"

Leer Lado

$Area = Lado * Lado$

Imprimir "El área del cuadrado es: "

Area

Fin Subprograma

RUTINA o subprograma: <Hallar-Area-Pentágono>

Inicio

PSEUDOCODIGO

Imprimir "Introduzca el lado del pentágono"

Leer Lado

Imprimir "Introduzca la apotema del pentágono"

Leer Apotema

$Area = (Lado * Apotema * 5) / 2$

Imprimir "El área del pentágono es: " Area

Fin Subprograma

## PROGRAMA 3

CARGAR UNA TABLA DESDE EL TECLADO (terminal).

Crear un programa que sea capaz de cargar los 10 nombres de frutas que más te gusten.

Inicio

Definicion de Datos

TABLA-FRUTAS para 10 FRUTAS: FRUTA(10)

Indice=1 'Indice para apuntar a cada elemento

FRUTATecleada= ' '

ContinuarSiNo="S"

Pseudocodigos (forma con DO WHILE y test antes )

Indice =1 'No es imprescindible, ya estaba inicializado

DO WHILE (Indice <= 10 AND ContinuarSiNo="S" )

Imprimir "TECLEA UN GASTO DE DIA"

Leer FRUTATecleada

FRUTA(Indice) = FRUTATecleada

Indice = Indice + 1

Imprimir "Si desea continuar pulse 'S' "

Leer ContinuarSiNo

END-DO

IF Indice > 10 Then

Imprimir "LA TABLA ESTA LLENA"

ELSE

Imprimir "AUN QUEDAN ELEMENTOS VACIOS"

END-IF

fin

# INSTRUCCIONES DE MANIPULACIÓN DE DATOS

## 1. VERBOS DE MANIPULACION DE DATOS

### FUNCIÓN:

Transferencia de los datos para su reorganización y transformación.

### MISIÓN DEL VERBO VERBO

\* TRANSFERENCIA DE DATOS > **MOVE**

\* INICIALIZACIÓN DE VARIABLES > **INITIALIZE**

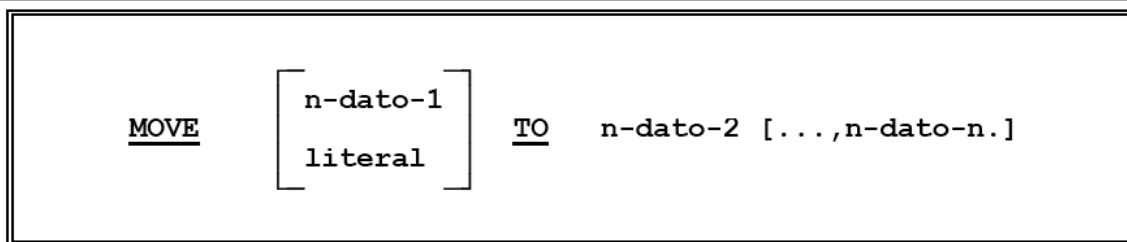
## 2. VERBO MOVE

### FUNCIÓN:

TRANSFERIR DATOS desde UNA ZONA de MEMORIA A OTRA. La instrucción MOVER implica “**copiar**” el dato de origen en el destino. El campo emisor siempre conserva el contenido.

La instrucción MOVE admite dos formatos básicos

### FORMATO 1:



El contenido de n-dato-1 o un literal es TRANSFERIDO a uno o más campos: n-dato-2...n-dato-n.



**ESQUEMA DE EJECUCION DE LOS MOVIMIENTOS:****AN:** Movimiento ALFANUMÉRICO:**NU:** Movimiento NUMÉRICO.**E :** Movimiento EDICIÓN.

TIPO DE MOVIMIENTO	ITEM RECEPTOR	ALINEACIÓN	RELLENO SI ES NECESARIO	TRUNCAMIENTO SI ES NECESARIO
ALFANUMÉRICO	Grupo (Campo compuesto)	A la izquierda del campo receptor	A la derecha con espacios	A la derecha
	Alfabético o alfanumérico	A la izquierda del campo receptor	A la derecha con espacios	A la derecha
NUMÉRICO	Decimal externo o decimal empaquetado	En punto decimal	A la izquierda y a la derecha con ceros	A la izquierda y a la derecha
EDICIÓN	Editado	En punto decimal	A la izquierda y a la derecha con ceros  (salvo que se supri- man)	A la izquierda y a la derecha

**MOVIMIENTOS PERMITIDOS:**

CAMPO EMISOR	CAMPO RECEPTOR
ALFABÉTICO	ALFABÉTICO Y ALFANUMÉRICO (AN)
ALFANUMÉRICO	ALFANUMÉRICO (AN)
NUMERICO DE CUALQUIER TIPO	NUMÉRICO DE CUALQUIER TIPO (NU)
	EDITADO (E)

\* Cualquier campo compuesto o grupo es alfanumérico por definición.

\* No se consideran otros movimientos posibles por considerarse inusuales.

**EFFECTO DE MOVIMIENTOS ALFANUMÉRICOS:**

EMISOR			RECEPTOR			
PIC	VALOR	Config.	PIC	VALOR	Config.	RESULTADO
X(4)	'A1C2'	C1F1C3F2	X(4)	'XT01'	E7E2F0F1	C1F1C3F2 'A 1 C 2'
X(2)	'B5'	C2F5	X(4)	'AAAA'	C1C1C1C 1	C2F54040 B 5
X(4)	'BBBB'	C2C2C2C2	XX	'AA'	C1C1	C2C2 'B B'
X	' '	40	X(4)	'TTTT'	E3E3E3E3	40404040 ' '
X(3)	'DDD'	C4C4C4	X(4)	'HHH2'	C8C8C8F2	40C4C4C4C ' D D D'

**A:** Blancos a la derecha.

**B:** trunca por la derecha.

**J:** Con cláusula **JUSTIFIED RIGHT**.

**77 CAMPO-RECEPTOR PIC X(4) JUST RIGHT.**

**EFFECTOS DE MOVIMIENTOS NUMÉRICOS:****EMISOR****RECEPTOR**

PIC	VALOR	Config.	PIC	VALOR	Config.	RESULTAD	
9 (4)	1234	F1F2F3F4	9 (4)	0000	FOFOFOFO	F1F2F3F4 1 2 3 4	
99	12	F1F2	9 (4)	1234	F1F2F3F4	F0F0F1F2 0 0 1 2	C
9 (4)	1234	F1F2F3F4	99	12	F1F2	F3F4 3 4	D
99V9	12.3	F1F2F3	99V9	00.0	F0F0F0	F1F2F3 1 2.3	
9V9	1.2	F1F2	99V99	00.00	F0F0F0F0	F0F1F2F0 0 1.2 0	
V99	.50	F5F0	9 (3)	12	F0F1F2	F0F0F0 0 0 0.	ByC
S9 (3)	123+	F1F2C3	9V99	5.55	F5F5F5	F3F0F0 3.0 0	EyD
9 (3)	123	F1F2F3	S99	55-	F5D5	F2C3 2 3 +	FyD

**B:** Trunca por la derecha**C:** Ceros a la izquierda.**D:** Trunca por la izquierda.**E:** Pierde signo**F:** Gana signo.

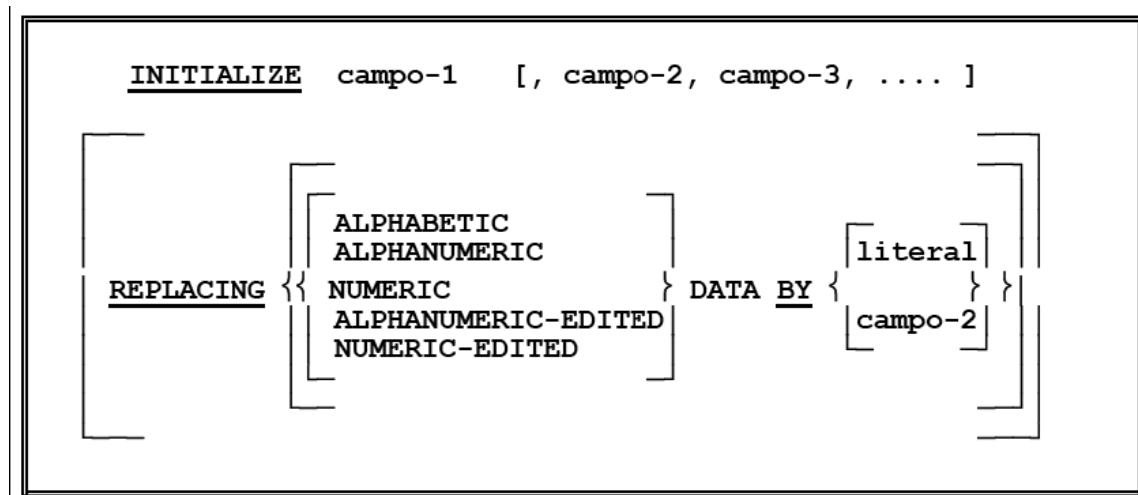
### 3. VERBO INITIALIZE

#### FUNCIÓN:

Esta instrucción permite **inicializar** cualquier variable, con el valor apropiado, con independencia de que el campo estuviese definido con la cláusula VALUE.

Su diferencia con la cláusula VALUE, consiste, en que con INITIALIZE se puede usar tantas veces como se desee, y en cualquier sitio dentro de la PROCEDURE, mientras que el valor asignado con el VALUE se introduce únicamente al comienzo del programa y se define en la WORKING.

#### FORMATO:



#### campo-1, campo-2, campo-3:

Campo **simple** o **compuesto** que se desea inicializar.

**REPLACING:** Permite iniciar **únicamente** los campos elementales que concuerdan con la clase especificada.

Si no se codifica REPLACING todos los campos NUMÉRICOS y de EDICIÓN se inicializan con CEROS y los ALFANUMÉRICOS a BLANCOS.

**DATA BY:** Siguiendo a esta cláusula se pone el valor con el que se va a inicializar campo-1. Se permite un literal o nombre de campo.

**Ejemplo:**

	A	B
4 6 7 8	12 16 20 24 28 32 36 40 44 48 52 56 60 64 72	
010	WORKING-STORAGE SECTION.	
020	*****	
020	77	CONTA-LIN PIC S9(4) VALUE +80 COMP-3.
030	77	CODIGO-RETORNO PIC XX VALUE 'SI'.
040	01	CAMPO-1.
050	02	ALFABETICO PIC A(5).
060	02	ALFANUMERICO PIC X(10).
070	02	NUMERICO PIC 9(6)V9(2).
080	02	ALFANU-EDIT PIC XX/XX/XXXX.
090	02	NUM-EDIT PIC ZZ.ZZ9,99.
100	PROCEDURE DIVISION.	
110	. . . . .	
120	INITIALIZE CONTA-LIN CODIGO-RETORNO CAMPO1.	
130	*	Se inicializan con "Ø" los campos numéricos y
140	*	con SPACES los campos alfanuméricos.
150		
160	INITIALIZE CONTA-LIN	
170	REPLACING NUMERIC DATA BY +50.	
180		
190	INITIALIZE CAMPO-1	
200	REPLACING	
210	ALPHABETIC DATA BY ALL 'T'	
220	ALPHANUMERIC DATA BY ALL '*'	
230	NUMERIC DATA BY 7,5	
240	ALFANUMERIC-EDIT DATA BY 27081990	
250	NUMERIC-EDIT DATA BY 11111,11.	

El contenido de CAMPO-1 después de la iniciación será:

TTTTT*****0000075027/08/199011.111,11
---------------------------------------

# LLAMADAS ENTRE PROGRAMAS: LINKAGE SECTION

## 1. CONCEPTO DE SUBPROGRAMACIÓN

La programación estructurada se basa, en la descomposición del proceso general en subprocesos más fáciles de solucionar y mantener.

La subprogramación, permite obtener programas más pequeños, fáciles y manejables además de hacer más cómodas las posibles modificaciones futuras. Además se evitan redundancias, ya que las tareas comunes a varios programas pueden ser realizadas por subprogramas, los cuales serán llamados para que realicen esas tareas.

Siempre se tendrá un **PROGRAMA PRINCIPAL**, ( “ **LLAMANTE** “ ), que será el que se encargue de llamar a **uno o más SUBPROGRAMAS**, ( “ **LLAMADO/S** “ ).

El programa PRINCIPAL inicia y controla la ejecución de todo el proceso.

Un subprograma podrá a su vez llamar a otros subprogramas, pero nunca al principal.

Cuando un programa llama a otro (**CALL**), el control se pasa al llamado para su ejecución y cuando ésta finaliza (**EXIT PROGRAM**), el control es devuelto de nuevo al programa llamante, para continuar la ejecución del mismo, a partir de la sentencia que efectuó la llamada.

Los subprogramas dependen para su ejecución del programa principal. Para ello es preciso incluir dentro de cada programa las **instrucciones de transferencia de control** oportunas, a fin de indicar al Sistema Operativo el momento en que cada programa ha de suspender su proceso para pasar el control a otro programa y cuáles son **los campos o variables que se pasan entre sí** para usar durante su ejecución.

Si los programas se compilan por separado, se producirán módulos objetos distintos, pero si se linkeditan juntos durante la ejecución, en la memoria principal estará cargado la unión del programa principal y todos los subprogramas.

[illegible]



**\* INSTRUCCION PROHIBIDA.****3. LINKAGE SECTION Y COMUNICACIÓN ENTRE PROGRAMAS****FUNCIÓN**

Definición en los Programas LLAMADOS o subprogramas de los CAMPOS coinciden tes con campos del Programa LLAMANTE.

**PECULIARIDADES DE LA LINKAGE SECTION:**

- « Es una sección de la DATA DIVISION, cuya utilización es OPCIONAL.
  - « Solamente se codifica en los SUBPROGRAMAS (en los programas llamados).
  - « Mantiene la misma estructura y características de la WORKING-STORAGE, excepto que la cláusula VALUE no está permitida.
  - « En la LINKAGE SECTION (del programa llamado), se definen todos los campos que el programa llamante va a permitir que sean compartidos por los dos programas. A través de estos campos ambos programas, podrán pasarse datos y devolvérselos una vez procesados.
  - « El compilador no reserva memoria para los campos definidos en esta sección, ya que en realidad, el sistema lo que hace es permitir a los subprogramas referenciar y utilizar los campos definidos por el programa principal o por el subprograma que le ha llamado.
  - « NO es necesario utilizar los mismos nombres, para los campos comunes en ambos programas.
- Tampoco es necesario definirlos en el mismo orden en el programa llamante y llamado. (Aunque tanto una cosa como otra puede ser aconsejables, por la claridad que pueden aportar).
- « También **es transcendental** que se respeten en la LINKAGE **las mismas PICTURES** de los campos definidos en el programa principal, es decir, la misma LONGITUD, el mismo TIPO y el mismo USAGE de los campos que comparten ambos programas.
- Para poder pasar campos de un programa a otro, además de la LINKAGE SECTION, se necesitan varias sentencias características de estos procesos:

CALL, ENTRY, EXIT PROGRAM, GOBACK y CANCEL, las cuales, son propias de la PROCEDURE DIVISION.

« El orden de los campos en la PROCEDURE DIVISION asociados a la sentencia CALL del programa llamante y a la ENTRY del programa llamado debe de ser el mismo.

		A	B															
4	6	7	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	72
010																		
020	*																	
020			.		.													
030			.		.													
050																		
020	*																	
060			.		.													
070			.		.													
090																		
020	*																	
100		77		CAMPO														
110		01		CAMPO-1.														
120			02	CAMPO-11														
130			02	CAMPO-12														
140		01		CAMPO-2.														
150			02	CAMPO-22														
160			02	CAMPO-23														
170																		

#### 4. DECLARACIONES DE LA PROCEDURE DIVISION.

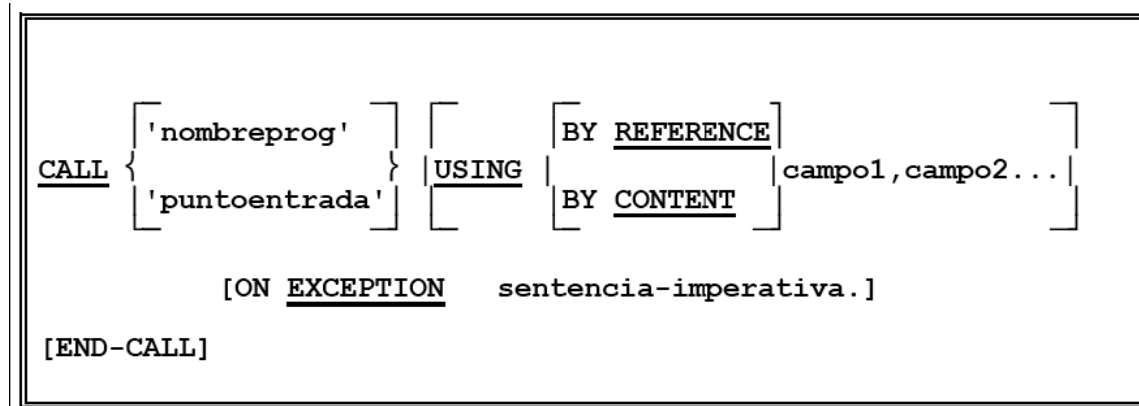
Las declaraciones características de la subprogramación son las siguientes:

- « CALL
- « ENTRY
- « EXIT PROGRAM
- « GOBACK
- « CANCEL

Los distintos formatos son analizados a continuación.

## 5. DECLARACIÓN CALL.

Permite la llamada a un programa desde otro programa. El programa llamante pasa el control del proceso al programa llamado.



Se llama al subprograma (**CALL nombreprog**) y se le permite usar (**USING**) el contenido de las variables (**campo1, campo2, etc.**) definidos en la DATA DIVISION del programa llamante y en la LINKAGE SECTION de **nombreprog**.

**Nombreprog:** Es el nombre del subprograma que se desea llamar, para que continúe el proceso.

Cuando se codifica en el formato **nombreprog** se sobreentiende que se desea comenzar justamente en el comienzo de la procedure del programa llamado. Y, nombreprog de la CALL coincide con el nombre del **PROGRAM-ID** del subprograma.

Cuando **no** se asocia, con la cláusula **USING**, es porque se desea pasar el control a un subprograma sin necesidad de pasarle información alguna.

Por ejemplo: Un programa principal que sea un menú y los subprogramas sean las distintas opciones a realizarse.

**Puntoentrada:** Se codifica puntoentrada cuando se desee pasar el control a otro programa, pero, a un punto distinto del comienzo de la procedure.

Tanto **nombreprog** como **puntoentrada** puede ser un literal alfanumérico o un nombre de campo que contenga el literal.

Un mismo subprograma puede ser llamado tantas veces y a tantos puntos de entrada distintos como se considere necesario.

**USING:** Lo más usual es que el programa LLAMANTE deba de pasar datos al LLAMADO, para ser elaborados y posteriormente devueltos al programa principal, para continuar con su proceso.

Si es este el caso, es necesario indicar el nombre de los campos detrás de la palabra reservada **USING**.

El **número de nombres**, detrás de USING debe de ser el mismo, tanto en el programa principal como en el subprograma. La correspondencia entre ellos es de uno a uno, y deben estar en la misma posición y tener la misma definición. Los campos pueden estar definidos en el programa principal dentro de la FILE SECTION o de la WORKING-STORAGE SECTION y a cualquier nivel.

La diferencia entre BY CONTENT y BY REFERENCE estriba en que los campos asociados a BY CONTENT que se pasan al subprograma, **sólo** pueden ser **consultados**, mientras que los asociados a BY REFERENCE, **podrán además ser modificados**. Por defecto asume BY REFERENCE.

**ON EXCEPTION:** Cada vez que llamamos a un programa vía **CALL** ese programa se carga en memoria. Puede ser conveniente, utilizar la opción **ON EXCEPTION**, para prevenir el caso en no haya suficiente espacio disponible en memoria para cargarlo, o no se encontrara el programa a cargar, o cualquier otro problema.

## 6. DECLARACION ENTRY

Establece un punto de entrada en un programa llamado.

### FORMATO-1:

```
ENTRY 'nombre-de-enlace' [ USING campo-1, campo-2... ]
```

### FORMATO-2:

```
PROCEDURE DIVISION [ USING campo-1 campo-2 ... ]
```

En cada subprograma, pueden existir uno o más puntos de entrada.

Cada punto de entrada, se identifica o asocia mediante un **nombre-de-enlace** o literal distinto, que se pone detrás del **ENTRY** y que debe coincidir con el literal que se puso en la sentencia **CALL** correspondiente dentro del programa principal.

Cuando se desea que el punto de entrada en el subprograma comience desde la primera instrucción de la PROCEDURE DIVISION, (se puede prescindir de **ENTRY 'nombre-enlace'** y especificar a continuación de las palabras PROCEDURE DIVISION, la opción **USING** seguida del nombre de los campos de datos que le va a pasar el programa llamante.

**Campo-1, campo-2, ...** etc, estarán definidos necesariamente en la LINKAGE SECTION del subprograma y a nivel 01 o 77.

## 7. DECLARACIONES EXIT PROGRAM Y GOBACK

Identifican el punto de bifurcación de retorno desde el programa llamado al programa que lo llamó.

En cada programa llamado debe existir al menos una declaración **EXIT PROGRAM** o **GOBACK** (la función de ambas es equivalente).

## 8. DECLARACIÓN CANCEL

Sirve para borrar de la memoria programas que fueran anteriormente llamados por una **CALL**.

**FORMATO:**

<p style="text-align: center;"><u><b>CANCEL</b></u>      'nombreprog' .</p>
---

Cuando se trabaja en “entornos pequeños”, es conveniente utilizar una instrucción **CANCEL** después de cada

**CALL** para que no haya desborda mientos de memoria.

## 9. ENLACE ESTÁTICO Y ENLACE DINÁMICO

Hay dos formas de enlazar programas, el enlace estático y el enlace dinámico.

Un **enlace estático** se produce cuando dos o más módulos objeto son unidos, por un programa montador de enlaces (link-editor) para producir **un único** módulo de carga ejecutable. En un enlace estático el o los subprogramas ocupan memoria en todo momento independientemente de que sean o no llamados.

En cambio, en un **enlace dinámico**, el o los subprogramas **no están unidos** al módulo de carga del programa de llamada, sino que están almacenados como un módulo de carga independientes en una biblioteca de módulos ejecutables. El subprograma no está inicialmente cargado en memoria junto con el programa de llamada. Es cargado únicamente si se le llama a través de la ejecución de la instrucción CALL correspondiente.

### **ENLACE ESTÁTICO VERSUS ENLACE DINÁMICO**

La principal ventaja del enlace dinámico estriba en que el subprograma no necesita ocupar memoria hasta que se le llame, y la principal desventaja es que se tarda más tiempo en ejecutar una CALL dinámica (puesto que se deberá cargar desde la librería donde resida)

El enlace estático se ejecuta más rápido que el dinámico, aunque requiere una mayor preparación, ya que el programa y el subprograma deben estar enlazados de antemano por el montador de enlace (o el equivalente). El enlace estático es más conveniente para subprogramas externos que tienden a ser plenamente utilizados.

La codificación de la sentencia CALL de ambos tipos de enlaces es idéntica excepto que, para implementar un enlace estático el nombre del subprograma que se quiere llamar (PROGRAM-ID) está colocado en un campo alfanumérico (generalmente en la WORKING-STORAGE SECTION) antes de que la instrucción CALL sea ejecutada.

El encontrar un **nombre de campo** en vez de un **literal alfanumérico** le indica al compilador que se trata de un enlace estático. El subprograma externo estará escrito exactamente de la misma forma para los dos tipos de llamadas.

```

EJEMPLOS: PROGRAMA PRINCIPAL.
  A  B
4 6 7 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 72
010 IDENTIFICATION DIVISION.
020 * =====
030 PROGRAM-ID. PRINCIPAL.
040 AUTHOR. ESTEBAN DIAZ-MINGO CARREÑO.
050 * *****
060 * * El Programa Principal, solicita la fecha de nacimiento y la *
070 * * pasa al Subprograma-1 para que averigüe la edad de la persona *
080 * * y después de responder, Pregunta si se ha acertado o no. *
090 * *****
100
110 * DATA DIVISION.
120 * =====
140 WORKING-STORAGE SECTION.
150 * -----
160 01 RESPUESTA-DEL-PROGRAMA.
170 02 FILLER PIC X(18) VALUE 'SI HAS NACIDO EN: '.
180 02 FECHA-DE-NACIMIENTO-DE-ENTRADA.
190 03 DIA PIC 9(2) VALUE 0.
200 88 DIA-OK VALUE 01 THRU 31.
210 03 FILLER PIC X VALUE '-'.
220 03 MES PIC 9(2) VALUE 0.
230 88 MES-OK VALUE 01 THRU 12.
240 03 FILLER PIC X VALUE '-'.
250 03 ANIO PIC 9(2) VALUE 0.
260 88 ANIO-OK VALUE 00 THRU 99.
270 02 FILLER PIC X(9) VALUE ' TIENES "'.
280 02 EDAD-A-CALCULAR PIC 9(2) VALUE 0.
290 02 FILLER PIC X(15) VALUE '" AÑOS. ¿OK? '.
300
310 01 CONFIRMACION.
320 02 FILLER PIC X(60) VALUE '** PULSA "S" (si) ó
330 - "N" (no) PARA SABER SI HE ACERTADO'.
340 02 RESPUESTA PIC X VALUE ' '.
350 88 EDAD-INCORRECTA VALUE 'N' 'n'.
360 88 RESPUESTA-EDECuada VALUE 'N' 'n' 'S' 's'.
370
380 PROCEDURE DIVISION.
390 * =====
400 PERFORM ACEPTAR-FECHA-DE-NACIMIENTO UNTIL
410 DIA-OK AND MES-OK AND ANIO-OK.
420 CALL 'SUBPROG1' USING FECHA-DE-NACIMIENTO-DE-ENTRADA
430 EDAD-A-CALCULAR.
440 DISPLAY RESPUESTA-DEL-PROGRAMA CONFIRMACION.
450 ACCEPT RESPUESTA.
460 PERFORM UNTIL RESPUESTA-ADECUADA
470 DISPLAY 'HA PULSADO=>' RESPUESTA 'Y HA DE SER N ó S.'
480 ACCEPT RESPUESTA.
490 IF EDAD-INCORRECTA
500 CALL 'LLAMADA2' USING RESPUESTA
510 ELSE
520 DISPLAY '**LOS ' EDAD-A-CALCULAR ' SON FASCINANTES**'.
530 STOP RUN.
540
550 ACEPTAR-FECHA-DE-NACIMIENTO.
560 DISPLAY 'TECLEE LA FECHA DE NACIMIENTO, CON EL FORMATO:'.
570 DISPLAY 'DD-MM-AA<== Y LE DIGO LA EDAD QUE TIENE.'.
580 ACCEPT FECHA-NACIMIENTO-DE-ENTRADA.
590 IF NOT DIA-OK OR NOT MES-OK OR NOT ANIO-OK
600 DISPLAY 'DD-MM-AA<== **** FECHA MAL INTRODUCIDA **'
610 DISPLAY '*****'.
620

```

```

EJEMPLOS: SUBPROGRAMA LLAMADO.
| A | B |
4 6 7 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 72
-----
010 IDENTIFICATION DIVISION.
020 * =====
030 PROGRAM-ID. SUBPROG1.
040 AUTHOR. ESTEBAN DIAZ-MINGO CARREÑO.
050 * *****
060 * Este Subprograma obtiene la fecha actual del ordenador, y a *
070 * partir de la fecha de nacimiento, que le pasa el Programa *
080 * Principal, calcula la edad de la persona. Además, si la *
080 * fecha actual, no es correcta, informa de ello. *
090 * *****
100
110 * DATA DIVISION.
120 * =====
140 WORKING-STORAGE SECTION.
150 * -----
170 01 FECHA-ACTUAL.
180 02 ANIO-ACTUAL PIC 99 VALUE ZERO.
190 02 MES-ACTUAL PIC 99 VALUE ZERO.
200 02 DIA-ACTUAL PIC 99 VALUE ZERO.
210
220 LINKAGE SECTION.
230 * -----
240 01 EDAD-SALIDA PIC 99.
250
260 01 FECHA-DE-NACIMIENTO-PASADA.
270 02 DIA-NACIM PIC 99.
280 02 FILLER PIC X.
290 02 MES-NACIM PIC 99.
300 02 FILLER PIC X.
310 02 ANIO-NACIM PIC 99.
320
330 01 RESPUESTA-NEGATIVA PIC X.
340
350
360 PROCEDURE DIVISION
370 * =====
380 USING FECHA-DE-NACIMIENTO-PASADA
390 EDAD-DE-SALIDA.
400
410 ACCEPT FECHA-ACTUAL FROM DATE.
420 COMPUTE EDAD-SALIDA = ANIO-ACTUAL - ANIO-NACIM.
430
440 IF ( MES-ACTUAL < MES-NACIM ) OR
450 ( MES-ACTUAL = MES-NACIM AND DIA-ACTUAL < DIA-NACIM )
460 SUBTRACT 1 FROM EDAD-SALIDA.
470 EXIT PROGRAM.
480 ENTRY 'LLAMADA2' USING RESPUESTA-NEGATIVA.
490 DISPLAY '¿ SI HA RESPONDIDO: ' RESPUESTA-NEGATIVA ' ?'.
500 DISPLAY 'POSIBLEMENTE LA FECHA ACTUAL DEL ORDENADOR SEA '
510 'INCORRECTA. ** COMPRUEBELO **'.
520 DISPLAY 'LA FECHA ACTUAL DEL ORDENADOR ES LA SIGUIENTE: '.
530 DISPLAY ' DIA= ' DIA-ACTUAL
540 ' MES= ' MES-ACTUAL
550 ' AÑO= ' ANIO-ACTUAL.
560 EXIT PROGRAM.
570 * GOBACK. (También sería válido).
580
590

```



# INTERFASE DE PROGRAMACIÓN DE ALTO NIVEL

## 1. RESTRICCIONES DEL COMPILADOR COBOL

En un programa COBOL **no** se podrá usar:

- En la ENVIRONMENT DIVISION: la INPUT-OUTPUT SECTION.
- En la DATA DIVISION: la FILE SECTION.
- En la PROCEDURE DIVISION, los verbos:

ACCEPT, CLOSE, DELETE, DISPLAY, MERGE, OPEN, READ, RERUN, REWRITE, START, WRITE. Nuestro programa se comunicará con el CICS, por medio de éste interfase, que se soporta para cuatro lenguajes diferentes: COBOL, ENSAMBLADOR, PL/I y RPGII.

## 2. BLOQUE DE EJECUCION DE LA INTERFASE (EIB)

Automáticamente el traductor copia la EIB. Durante la ejecución las EIB sin inicializadas por el programa de ejecución de la interfase. El programa de aplicación puede recuperar información del EIB usando el nombre de campo apropiado. Estos nombres de campos de la EIB son palabras reservadas.

La información que contienen los campos más usados es:

- **EIBTIME**; La hora en que fue inicializada una tarea (hhmmss).
- **EIBDATE**; La fecha en que fue inicializada una tarea (YYDDD).
- **EIBTRNID**; El código de transacción asociado a esa tarea.
- **EIBTASKN**; El número de tarea que el CICS ha asignado a una tarea.
- **EIBTRMID**; El identificador del terminal asociado a esa tarea.
- **EIBCPOSN**; La posición del cursor al realizarse una transmisión desde un terminal.
- **EIBCALEN**; La longitud de la COMMAREA que se recibe en este programa.
- **EIBAUD**; Contiene un código que identifica el tipo de tecla pulsada.
- **EIBFN**; Código de la última función usada.
- **EIBRCODE**; Código de retorno del CICS después de ejecutar un mandato.
- **EIBDS**; El identificador simbólico del último conjunto de datos en una petición al Control de Ficheros.
- **EIBREQUID**; El identificador de petición para una petición al Control de Intervalos.

- **EIBERR**; Es un switch de error, que indica si está activo que se ha producido un error en un mandato CICS.

La codificación en COBOL de los campos de este bloque es:

01 DFHEIBLK.

02 EIBTIME PIC S9(7) COMP-3. (Fecha)

02 EIBDATE PIC S9(7) COMP-3. (Hora)

02 EIBTRNID PIC X(4). (Cod. de TRN. asoci. a la tarea)

02 EIBTASKN PIC S9(7) COMP-3. (Nº de la tarea)

02 EIBTRMID PIC X(4). (Terminal asociado a la tarea)

02 DFHEIGDI PIC S9(4) COMP. (Reservado por el cics)

02 EIBCPOSN PIC S9(4) COMP. (Posición del cursor en la entrada)

02 **EIBCALEN** PIC S9(4) COMP. (Long. comarea del último mand.)

02 **EIBAID** PIC X. (Valor de la TECLA pulsada)

02 EIBFN PIC X(2). (Ident. del último mand. cics ejec.)

02 EIBRCODE PIC X(6). (Código del últi. mand. cics ejec.)

02 EIBDS PIC X(8). (Nombre de fichero usado en ...)

02 EIBREQUID PIC X(8).

02 EIBRSRCE PIC X(8).

02 EIBSYNC PIC X.

02 EIBFREE PIC X.

02 EIBRECV PIC X.

02 EIBFILO2 PIC X.

02 EIBATT PIC X.

02 EIBEOC PIC X.

02 EIBFMH PIC X.

02 EIBCOMPL PIC X.

02 EIBSIG PIC X.

02 EIBCONF PIC X.

02 EIBERR PIC X.

02 EIBERCD PIC X.

02 EIBSYNRB PIC X.

02 EIBNODAT PIC X.

02 EIBSYNRB PIC X.

02 EIBNODAT PIC X.

02 **EIBRESP** PIC S9(8) COMP. (Nº de la condición de ERROR)

02 EIBRESP2 PIC S9(8) COMP. (Más infor. Sobre el error)

### 3. FORMATO DE LOS MANDATOS

El formato general de un mandato CICS es:

**EXECUTE CICS función opción**

**opción(argumento)**

•  
•  
•  
•

(Podemos codificar EXEC en lugar EXECUTE).

Donde:

- “función” Describe la operación requerida. Sólo puede haber una función asociada con cada mandato CICS.
- “opción” Describe cada una de las facilidades opcionales utilizables para cada función.

Algunas opciones van seguidas por un argumento entre paréntesis, otras no.

Las opciones pueden ser incluidas en cualquier orden.

- “argumento” Es el valor de un dato o el nombre de un área de datos.

Si éste área no está definida en nuestro programa irá codificado entre comillas, si ya lo está en el programa, irá sin ellas.

#### 4. MANDATO DE CONDICIONES DE EXCEPCIÓN

**EXEC CICS HANDLE CONDITION condición(etiqueta)**  
**condición(etiqueta)**

·  
·  
·

La condición es el identificador de error. Puede haber hasta 16 condiciones en un mandato HANDLE.

La etiqueta es el nombre del punto del programa a donde bifurca en el caso de cumplirse esta condición.

Ejemplo:

```
EXEC CICS HANDLE CONDITION ENDFILE(FIN-FICHERO)
ERROR(OTRO-FICHERO)
END-EXEC
```

Este mandato debe codificarse previamente al mandato CICS al que afecte; de forma que en el primer paso de traducción es transformado, en Cobol, en GO TO DEPENDING ON o ALTER, afectando a todos los mandatos CICS que hubiera por detrás hasta el siguiente mandato HANDLE CONDITION existente.

#### 5. MANDATO IGNORE

El mandato CICS IGNORE se encarga de anular el tratamiento de una determinada condición excepcional para todos los mandatos CICS posteriores a él.

Formato:

**EXEC CICS IGNORE CONDITION condición**

Si quisiéramos anular todas las condiciones excepcionales de un determinado mandato lo haremos con la opción NOHANDLE en ese mandato.

Formato:

**EXEC CICS .....**

.....

.....

**NOHANDLE**

## 6. CONTROL DE LAS CONDICIONES DE EXCEPCION USANDO LA OPCION "RESP"

El problema de la anterior forma de trabajar es que se obtenían programas poco estructurados. A partir de la versión 1.7 todos los mandatos CICS soportan una opción nueva, RESP. Esta opción permite el control de las condiciones excepcionales de forma estructurada.

### - Control con HANDLE CONDITION

WORKING-STORAGE SECTION.

.

.

.

PROCEDURE DIVISION.

.

.

.

EXEC CICS HANDLE CONDITION

MAPFAIL (MAPA-VACIO)

ERROR (CANCELAR)

END-EXEC.

EXEC CICS SEND MAP ('MAPA')

MAPSET ('FYC100F')

ALARM

ERASE

FREEKB

FRSET

END-EXEC.

.

.

.

**- Control con la opción RESP.**

WORKING-STORAGE SECTION.

77 RESPUESTA PIC S9(8) COMP.

.

.

.

PROCEDURE DIVISION.

EXEC CICS HANDLE CONDITION

ERROR (CANCELAR)

END-EXEC.

.

.

.

EXEC CICS SEND MAP ('MAPA')

MAPSET ('FYC100F')

ALARM

ERASE

FREEKB

FRSET

RESP (RESPUESTA)

END-EXEC.

IF RESPUESTA = DFHRESP (MAPFAIL)

PERFORM MAPA-VACIO.

.

.

.

Aunque usemos la opción RESP conviene tener, al principio de la PROCEDURE DIVISION, un mandato HANDLE CONDITION para controlar un posible error general no contemplado.

# TECLAS



## 1 TECLAS DE PANTALLA

### CLEAR

Elimina el contenido total de la pantalla.

El contenido se puede recuperar, presionando la tecla PA2.

### ERASE EOF

Elimina el contenido o parte del contenido del campo a partir de donde esté el cursor.

El contenido se puede recuperar, presionando la tecla PA2.

### RESET CURSOR

Envía el cursor a la primera posición del primer campo del usuario definido en el panel.

### TAB

Envía el cursor a la primera posición del próximo campo del usuario, (en sentido de arriba abajo y de izquierda a derecha).

### Bloq

Fija 'letras mayúsculas', quedan disponibles las funciones 'de arriba' de las teclas standard con doble función.



Igual que la anterior, pero sólo mientras se fija la tecla. Elimina la función de la anterior.

### RESET

Restaura efectos de otras teclas, tales como INS MODE, etc...



Mueve el cursor de derecha a izquierda.

### RETURN

Envía el cursor a la primera posición del próximo campo del usuario, (en sentido de arriba abajo).

**ENTER**

Libera y envía al ordenador central los mandatos del usuario.

**INSERT**

Facilita la inserción de nuevos caracteres **MODE** entre los ya escritos en una línea. Su efecto se neutraliza por medio de la tecla **RESET**.

**SUPRIMIR o DEL**

Elimina caracteres que se encuentren a continuación del cursor, uno a uno.

**PA1**

Cancela comandos de TSO en ejecución.

También es útil en consultas de IMS

**PA2**

Es útil en consultas de IMS.

**FLECHA ARRIBA**

Desplaza el cursor, posición a posición, hacia arriba.

**FLECHA ABAJO**

Desplaza el cursor, posición a posición, abajo.

**FLECHA IZQUIER.**

Desplaza el cursor, posición a posición, hacia la izquierda.

**FLECHA DERECHA**

Desplaza el cursor, posición a posición, hacia la derecha.

**2 TECLAS PROGRAMABLES**

'ISPF' facilita 12 funciones que, para mayor comodidad del usuario, se asignan a las teclas que a continuación se describirán. Las 6 primeras teclas, disponen de doble función. Para el usuario de 'ISPF' son aplicables las funciones de la parte baja de las teclas. Las funciones de programa suelen estar asignadas de la forma que aquí se indica. En todo caso, el usuario podría modificar, si lo estimara conveniente, estas asignaciones de acuerdo con lo indicado en la opción 0.3

**PF1****‘HELP’**

Facilita información adicional sobre mensajes de condición de error o sobre opciones y mandatos ‘ISPF’.

**PF2****‘SPLIT’**

Parte la pantalla en dos permitiendo disponer en el mismo Terminal de dos pantallas de trabajo.

**PF3****‘END’**

Origina el retorno al panel anterior.

**PF4****‘PRINT’**

Graba el contenido de la pantalla actual en el fichero ‘spf(n).list’ para que posteriormente pueda ser impreso.

**PF5****‘FIND’**

Reitera la acción del mandato ‘FIND’, utilizando en distintas opciones, como se verá más adelante.

**PF6****‘CHANGE’**

Reitera la acción del mandato ‘CHANGE’ utilizado en la opción 2 (edit), como se verá más adelante.

**PF7****‘UP’**

Visualiza información situada por arriba de la actual de pantalla. Esta función actúa de acuerdo con lo escrito en SCROLL.

**PF8****‘DOWN’**

Visualiza información situada por debajo de la actual de pantalla. Esta función actúa de acuerdo con lo escrito en SCROLL.

**PF9****‘SWAP’**

Desplaza el cursor desde una pantalla a otra cuando la función SPLIT esta activa.

**PF10****‘LEFT’**

Visualiza información situada a la izquierda de la actual de pantalla. Esta función actúa de acuerdo con lo escrito en SCROLL.

**PF11****‘RIGHT’**

Visualiza información situada a la derecha de la actual de pantalla. Esta función actúa de acuerdo con lo escrito en SCROLL.

**PF12****‘RETURN’**

Origina el retorno al menú primario u a otra opción mediante el signo ‘=’ y el numero de la misma.

NOTAS PRELIMINARES.....	2
DEFINICIÓN DE CONCEPTOS.....	11
OTRAS FUNCIONES DEL COMPILADOR: .....	12
DEFINICIÓN DE DATOS.....	14
FORMATO GENERAL DE DEFINICIÓN DE CAMPOS.....	15
NORMAS DE CONSTRUCCION DE NOMBRES.....	16
CAMPOS COMPUESTOS.....	18
SPACE / SPACES.....	31
HIGH-VALUE/S: .....	32
LOW-VALUE/S:.....	32
ALL 'carácter/es': .....	33
LENGUAJE COBOL .....	37
REPRESENTACIÓN DE CARACTERES .....	37
PALABRAS COBOL RESERVADAS .....	37
Representación de Caracteres EBCDIC .....	38
Hoja de Codificación en COBOL .....	45
COPY .....	54
PERFORM .....	73
SPECIAL-NAMES.....	80
DECIMAL-POINT IS COMMA.....	80
INPUT-OUTPUT SECTION .....	80
PÁRRAFO FILE-CONTROL .....	80
INTRODUCCIÓN A LOS DATOS .....	81
DATA DIVISION .....	81
SECCIONES DE LA DATA DIVISION .....	81
FUNCIÓN.....	82
FUNCIONES Y PROCEDIMIENTOS .....	90
Variables Globales y Locales.....	92
FUNDAMENTOS DE FICHEROS VSAM Y MANDANTOS DEL AMS .....	105
INSTRUCCIONES DE ENTRADA / SALIDA .....	112
STOP RUN .....	142
ERRORES DE "EJECUCIÓN" .....	143
ARCHIVOS SECUENCIALES .....	150
ARCHIVOS INDEXADOS Y RELATIVOS .....	150
ANIMA.BAT NOMBRE-PROGRAMA (SIN EXTENSIÓN).....	152
F1 (Help) Ayuda.....	152
PROGRAMACIÓN CON TABLAS O ARRAYS.....	157
CARACTERÍSTICAS DE ENTORNO. ENVIRONMENT DIVISION.....	173
FILE STATUS.....	178
PROGRAMACIÓN CON CURSORES .....	180
PROCEDURE DIVISIÓ.....	191
TIPOS DE INSTRUCCIONES DE ENTRADA/SALIDA .....	197
COBOL EN PC.....	203
INSTRUCCIONES DE MANIPULACIÓN DE DATOS .....	207
LLAMADAS ENTRE PROGRAMAS: LINKAGE SECTION .....	214
TECLAS.....	232

