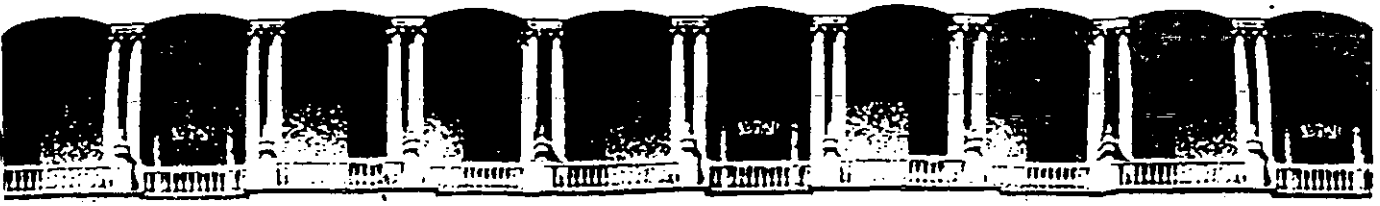


FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

METODOLOGIA DE LA PROGRAMACION

MATERIAL DIDACTICO

NOVIEMBRE, 1992.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D.F. Tel.: 521-40-20 Apdo. Postal M-2285

La programación de computadores

1.0. INTRODUCCION

Muchas personas piensan que un computador puede realizar tareas o trabajos de complejidad superior a una inteligencia humana. La realidad es que un computador no tiene ninguna inteligencia. No olvidemos que no es más que una máquina creada por el hombre y, por tanto, no podrá realizar una tarea que no haya sido previamente determinada por él.

Un computador, en general, sólo es capaz de realizar tres operaciones básicas:

1. Sumar, restar, multiplicar y dividir dos valores numéricos.
2. Comparar dos valores (comprobar si son iguales, si el primero es mayor que el segundo, etc.; estos valores pueden ser numéricos o alfabéticos).
3. Almacenar o recuperar información.

Con estas pocas operaciones utilizadas y combinadas de forma adecuada, mediante lo que llamamos programa de computador, se pueden llegar a realizar tareas increíblemente complejas que aporten la solución a un determinado problema, ya sea de gestión, técnico o de cualquier otro tipo.

La potencia de cálculo de un computador se deriva de las características físicas que posee:

Las características citadas provienen de los componentes electrónicos que conforman un computador: velocidad de conmutación de circuitos electrónicos, rapidez de transmisión de señales eléctricas, fiabilidad de los circuitos y gran capacidad de almacenamiento de informaciones en un reducido espacio físico.

Nuestro objetivo es, para un problema dado, diseñar una solución que pueda ser realizada por un computador. Para ello necesitaremos, en primer lugar, un lenguaje o notación para expresar la solución obtenida. Tal solución deberá estar adaptada a las particularidades del computador, aunque en una primera fase, para su diseño, podremos utilizar una notación intermedia entre el lenguaje natural y el del computador; pero, posteriormente, será preciso escribirla en un lenguaje comprensible por la máquina, como, por ejemplo, en **BASIC, COBOL** o **Pascal.**

1.1. FASES DEL DISEÑO Y PUESTA A PUNTO DE UN PROGRAMA

El proceso que se sigue desde el planteamiento de un problema, hasta que se tiene una solución instalada en el computador y lista para su ejecución, se compone de varias fases agrupadas en dos bloques bien diferenciados:

1.1.2.1. Fase de edición

Escritura del **programa fuente** a partir de las hojas de codificación en la memoria del computador, grabándolo en algún soporte permanente. Se hace con la ayuda de un programa del sistema llamado **editor.**

1.1.2.2. Fase de compilación

Traducción del programa fuente a lenguaje máquina cuyo resultado es el **programa objeto.** Para ello se dispone de programas **compiladores** o **intérpretes,** que además comprueban la correcta sintaxis del programa.

1.1.2.3. Fase de montaje

En los programas compilados es necesario añadir al programa objeto algunas rutinas del sistema o, en algunos casos, subprogramas externos que se hayan compilado separadamente. De ello se encarga el programa **montador (linker).**

1.1.2.4. Fase de ejecución

Consiste en probar el programa con datos de prueba, para asegurar el correcto funcionamiento del mismo.

1.2. CARACTERISTICAS DE LOS PROGRAMAS

Para un determinado problema se pueden construir diferentes algoritmos de resolución o programas. La elección del más adecuado se debe basar en una serie de reglas que adquieren gran importancia a la hora de evaluar el coste de su diseño y mantenimiento.

Las características generales que debe reunir un programa son las siguientes:

1.2.1. LEGIBILIDAD

Ha de estar escrito de tal forma que facilite su lectura y comprensión.

1.2.2. PORTABILIDAD

Su diseño debe permitir la codificación en diferentes lenguajes de programación, así como su instalación en diferentes sistemas.

1.2.3. MODIFICABILIDAD

Ha de facilitar su mantenimiento, esto es, las modificaciones y actualizaciones necesarias para adaptarlo a una nueva situación.

1.2.4. EFICIENCIA

Se deben aprovechar al máximo los recursos del computador, minimizando la memoria utilizada y el tiempo de proceso o ejecución.

1.2.5. MODULARIDAD

Ha de estar subdividido en bloques o módulos, cada uno de los cuales realizará una parte del conjunto del trabajo.

1.2.6. ESTRUCTURACION

Debe cumplir las reglas de la "Programación estructurada" para facilitar la verificación, depuración y mantenimiento del programa.

1.3. OBJETOS DE UN PROGRAMA: CONSTANTES Y VARIABLES

Son objetos de un programa todos aquellos manipulados por las instrucciones. Mediante ellos, en un programa podremos realizar el almacenamiento de los datos y de los resultados de las distintas operaciones que intervienen en la solución del problema.

1.3.1. ATRIBUTOS DE UN OBJETO

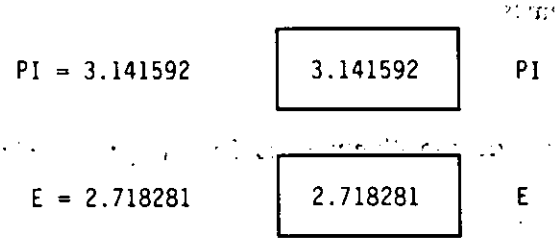
Todo objeto tiene tres atributos:

- Nombre: Es el identificador del mismo.
- Tipo: Conjunto de valores que puede tomar.
- Valor: Elemento del tipo que se le asigna.

1.3.2. CONSTANTES

Son objetos cuyo valor permanece invariable a lo largo de la ejecución de un programa. Una constante es la denominación de un valor concreto, de tal forma que se utiliza su nombre cada vez que se necesita referenciarlo.

Ejemplos típicos de constantes:



1.4.3.2. Operador AND

A	B	A AND B
F	F	F
F	C	F
C	F	F
C	C	C

Siendo A y B expresiones booleanas.

1.4.3.3. Operador OR

A	B	A OR B
F	F	F
F	C	C
C	F	C
C	C	C

Siendo A y B expresiones booleanas.

1.4.4. ORDEN DE EVALUACION DE LOS OPERADORES

Los operadores de una expresión se evalúan, en general, según el siguiente orden:

1.° Paréntesis (comenzando por los más internos).
2.° Potencias.
3.° Productos y divisiones.
4.° Sumas y restas.
5.° Concatenación.
6.° Relacionales.
7.° Negación.
8.° Conjunción.
9.° Disyunción.

La evaluación de operadores de igual orden se realiza siempre de izquierda a derecha.

Este orden de evaluación tiene algunas modificaciones en determinados lenguajes de programación.

Ejemplos: Evaluar las siguientes expresiones:

A:
```
((3 + 2)^2 - 15) / 2 * 5
   (5^2 - 15) / 2 * 5
    (25 - 15) / 2 * 5
        10 / 2 * 5
            5 * 5
              25
```

B:
```
5 - 2 > 4 AND NOT 0.5 = 1 / 2
5 - 2 > 4 AND NOT 0.5 = 0.5
  3 > 4 AND NOT 0.5 = 0.5
FALSO AND NOT 0.5 = 0.5
FALSO AND NOT CIERTO
FALSO AND FALSO
FALSO
```

Diagramas de flujo (flowchart)

2.0. INTRODUCCION

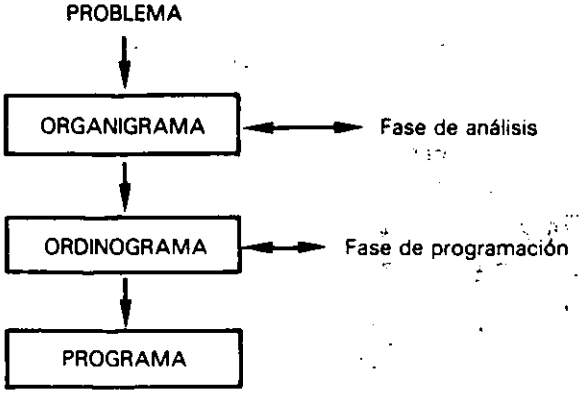
Durante el diseño de un programa, y en sus fases de análisis y programación, surge la necesidad de representar de una manera gráfica los flujos que van a seguir los datos manipulados por el mismo, así como la secuencia lógica de las operaciones para la resolución del problema.

Esta representación gráfica debe tener las siguientes cualidades:

- **Sencillez** en su construcción.
- **Claridad** en su comprensión.
- **Normalización** en su diseño.
- **Flexibilidad** en sus modificaciones.

En la práctica se suelen utilizar indistintamente los términos **diagrama de flujo, organigrama** y **ordinograma** para referenciar cualquier representación gráfica de los flujos de datos o de las operaciones de un programa. Es importante diferenciarlos porque no corresponden a las mismas fases del diseño de los programas. Aunque utilicen algunos símbolos comunes, el significado de éstos no es el mismo.

La secuencia de aparición de los diagramas de flujo se puede representar de la siguiente forma:



2.1. DIAGRAMAS DE FLUJO DEL SISTEMA

También se denominan **organigramas.** Representan gráficamente el flujo de datos e informaciones que maneja un programa.

En los casos de aplicaciones que comprenden más de un programa, se realizará un organi-

grama para cada uno de ellos, y es conveniente realizar uno general que englobe todo el conjunto.

Un organigrama debe reflejar:

— Los soportes de los datos.
— Los nombres de los programas.
— Los soportes de los resultados.
— El flujo de los datos.

En su representación se siguen las siguientes reglas:

— En el centro del organigrama figurará el símbolo de proceso correspondiente al programa.
— En la parte superior, los soportes de entrada.
— En la parte inferior, los soportes de salida.
— Al mismo nivel que el símbolo de proceso y a ambos lados, los soportes de entrada y salida.

Estas reglas estarán a su vez supeditadas a una presentación clara y precisa.
Los símbolos utilizados en la confección de organigramas son:

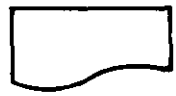
2.1.1. SIMBOLOS DE SOPORTE

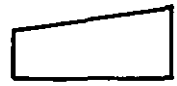
Representan los soportes físicos donde se encuentran los datos de entrada y donde van a ser registrados los resultados.

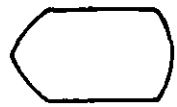
E = Soporte de entrada.
S = Soporte de salida.
E/S = Soporte de entrada y salida.

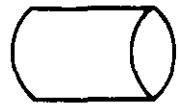

TARJETA PERFORADA (E/S)

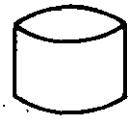

CINTA DE PAPEL (E/S)


IMPRESORA (S)


TECLADO (E)


PANTALLA (S)


TAMBOR MAGNETICO (E/S)


DISCO MAGNETICO (E/S)

DISCO MAGNETICO (E/S)

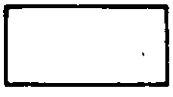
CINTA MAGNETICA (E/S)

DISCO FLEXIBLE (E/S)

CINTA ENCAPSULADA (E/S)

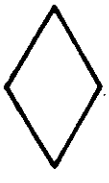
SOPORTE GENERICO (E)

2.1.2. SIMBOLOS DE PROCESO



PROCESO

OPERACION AUXILIAR

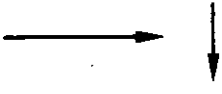


CLASIFICACION DE ARCHIVOS

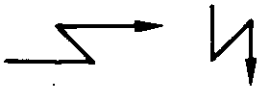
FUSION DE ARCHIVOS

PARTICION DE ARCHIVOS

2.1.3. LINEAS DE FLUJO



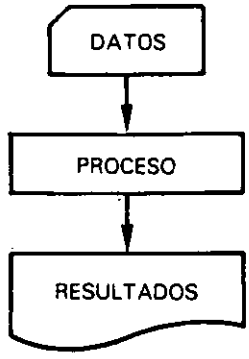
DIRECCION DEL FLUJO



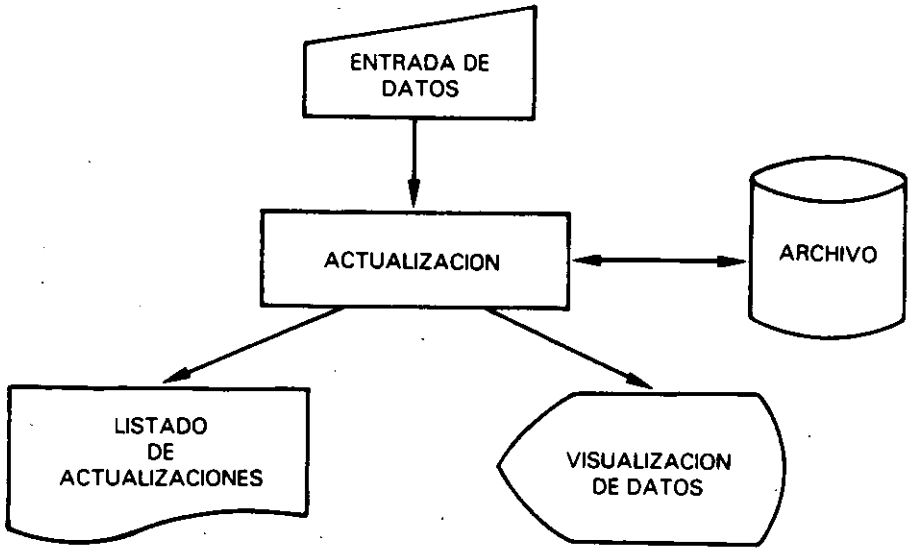
LINEAS DE TELEPROCESO

Ejemplos:

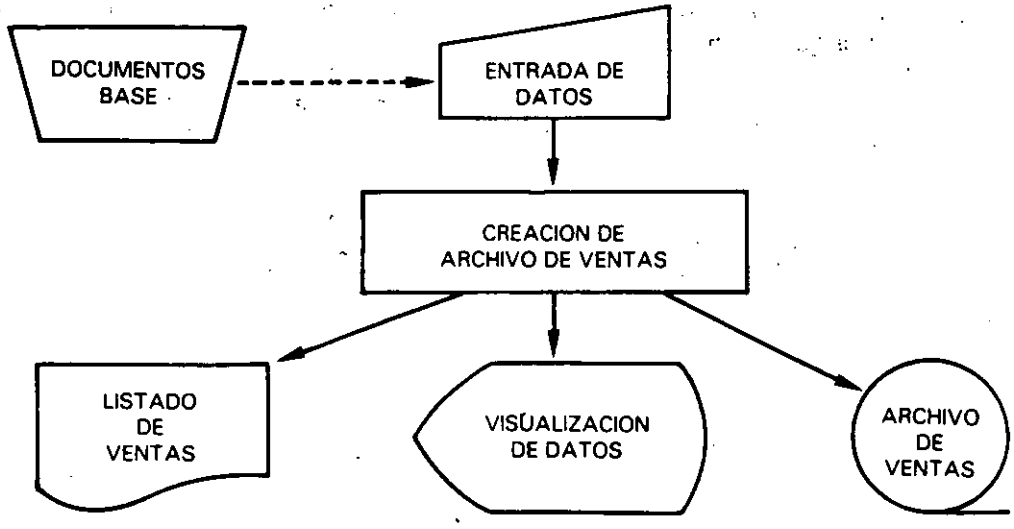
El organigrama de una aplicación que tome números perforados en fichas y realice alguna operación con ellos, dando los resultados por impresora, es:



El organigrama de una aplicación de actualización de un archivo soportado en disco, con entrada de datos por teclado, consulta de datos por pantalla y salida de datos modificados por impresora es:



El organigrama de una aplicación para crear un archivo de ventas en cinta, extraídas de un determinado documento base en papel, con un listado por impresora de dichas ventas y consultas por pantalla es:



2.2. DIAGRAMAS DE FLUJO DEL PROGRAMA

También se denominan **ordinogramas**. Representan gráficamente la secuencia lógica de las operaciones en la resolución de un problema, por medio de un programa de computador.

En la *fase de programación*, el programador crea para cada programa un ordinograma, a partir del cual realiza la codificación en lenguaje de programación.

Es necesario decir que el empleo de ordinogramas es una técnica de diseño de programas y no es la única, por lo tanto no es de obligado uso.

Resulta muy práctico, a nivel de metodología de diseño de programas, que se utilice una técnica general e independiente de los lenguajes de programación, puesto que los distintos lenguajes de que disponemos poseen sus propias características y restricciones, aunque suelen seguir una lógica similar.

Una vez diseñado un ordinograma que representa gráficamente un programa y sabiendo en qué lenguaje se va a codificar, lo único que necesitamos saber es la técnica de paso del ordinograma al lenguaje, que en la mayoría de los casos es bastante sencilla.

Hace tiempo que se mantiene una polémica entre los diseñadores de programas acerca de la utilidad de los diagramas de flujo. Unos manifiestan que su uso es innecesario, mientras que otros afirman la conveniencia de documentar todo programa con su ordinograma.

Es indudable que existen técnicas de diseño más modernas y adecuadas que los diagramas de flujo, como son las técnicas de diseño descendente y de programación estructurada; no obstante, la sencillez de éstos justifica su utilización como una primera técnica, y sobre todo como documentación para el posterior mantenimiento de los programas.

Un ordinograma debe reflejar:

— El comienzo del programa.
— Las operaciones.
— La secuencia en que se realizan.
— El final del programa.

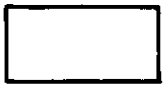
En la representación de ordinogramas, es conveniente seguir las siguientes reglas:

— El comienzo del programa figurará en la parte superior del ordinograma.
— Los símbolos de comienzo y fin deberán aparecer una única vez, utilizando el símbolo de parada (STOP) para representar cualquier otro tipo de interrupción o finalización.
— El flujo de las operaciones será, siempre que sea posible, de arriba a abajo y de izquierda a derecha, en cuyo caso pueden omitirse las puntas de flecha.
— Se debe guardar una cierta simetría en la representación de bifurcaciones y bucles, así como en el conjunto total del ordinograma.
— Se evitarán siempre los cruces de líneas de flujo utilizando conectores.
— El uso de comentarios estará restringido al mínimo imprescindible; al contrario que en la codificación, en la que son mucho más recomendables.
— Si en un ordinograma se ha separado una parte de otra por medio de un conector, las posibles conexiones que puedan aparecer desde esta última a la anterior, se harán igualmente con conectores, evitando el uso de líneas de flujo directas.

Los símbolos utilizados en la confección de ordinogramas son:

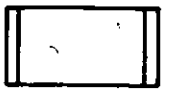
2.2.1. SIMBOLOS DE OPERACION

TERMINAL (INICIO, FIN, PARADA)



OPERACION EN GENERAL

OPERACION DE E/S EN GENERAL



SUBPROGRAMA

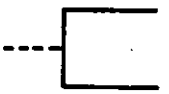
MODIFICACION DE PROGRAMA O
INICIALIZACION EN DECLARACIONES

OPERACION MANUAL

El símbolo de OPERACION DE E/S EN GENERAL, indica que la misma se realiza desde un dispositivo estándar de E/S.

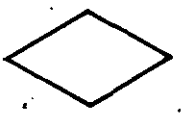
En la práctica, se utilizan los símbolos de soporte de organigramas para los ordinogramas en los casos que es necesario especificar el tipo de soporte utilizado.

2.2.2. SIMBOLO DE COMENTARIOS



COMENTARIO

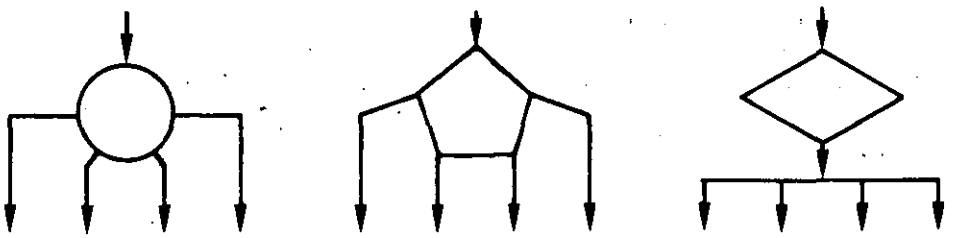
2.2.3. SIMBOLOS DE DECISION



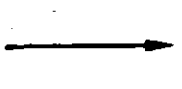
DECISION (2/3 SALIDAS)

DECISION MULTIPLE DE N SALIDAS

Se utilizan indistintamente el círculo, el polígono de N + 1 lados y el rombo. Por ejemplo, para 4 salidas:



2.2.4. LINEAS DE FLUJO



DIRECCION DEL FLUJO

2.2.5. SIMBOLOS DE CONEXION

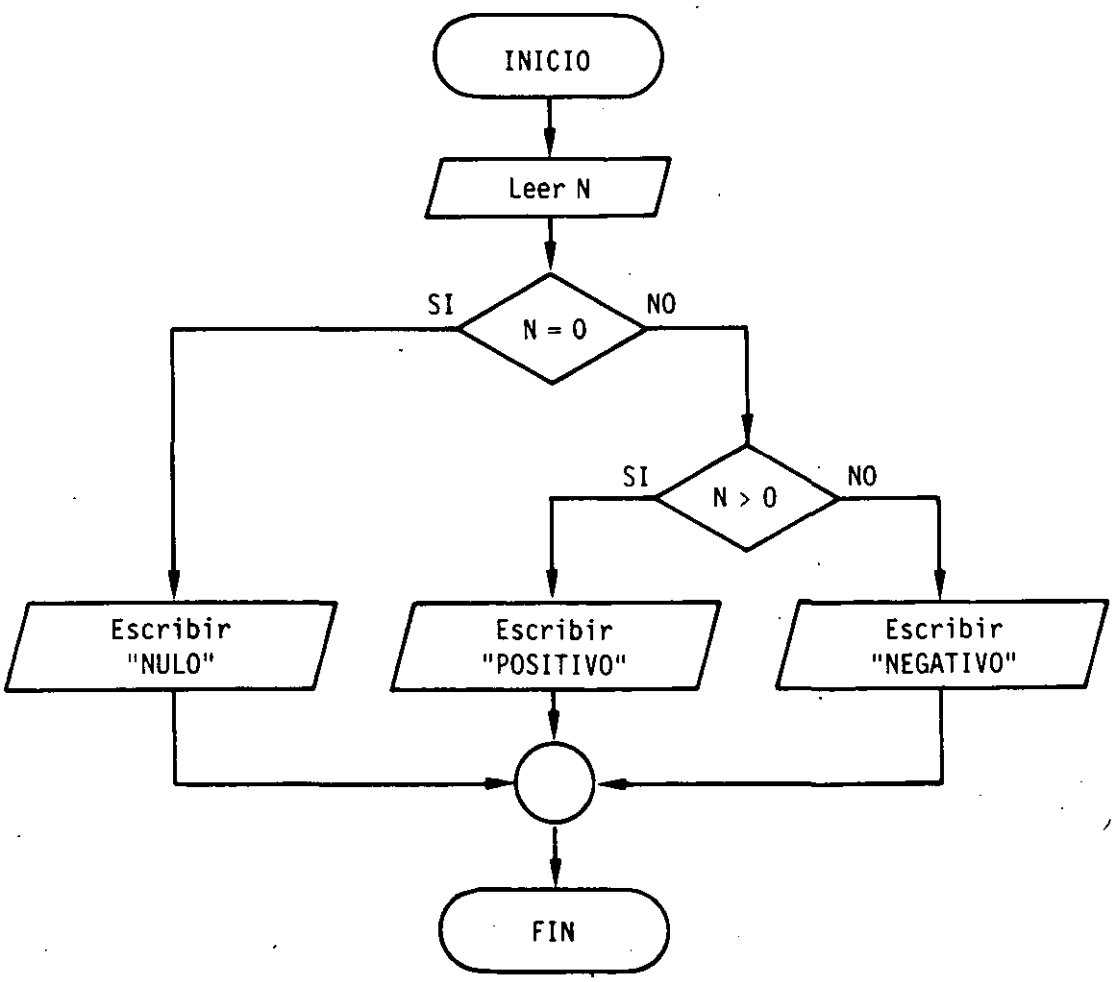
REAGRUPAMIENTO

CONECTOR EN LA MISMA PAGINA

CONECTOR EN DISTINTA PAGINA

Ejemplo:

Programa que lee un número del dispositivo estándar de entrada y comprueba e imprime en el dispositivo estándar de salida si dicho número es nulo, positivo o negativo.



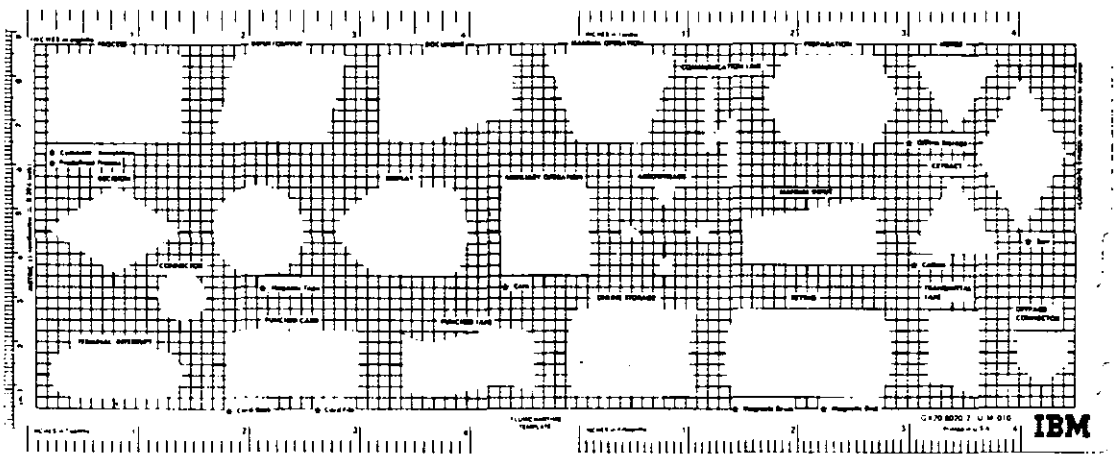
2.3. PLANTILLAS Y NORMALIZACION

El conjunto de símbolos gráficos utilizados en los diagramas de flujo necesitan estar normalizados para facilitar el intercambio de documentación entre el personal informático.

Al igual que sucede con los lenguajes de programación, conjunto de caracteres, protocolos de comunicación, etc., distintos organismos y fabricantes han dictado normas para la realización de diagramas de flujo, variando ligeramente de unos a otros.

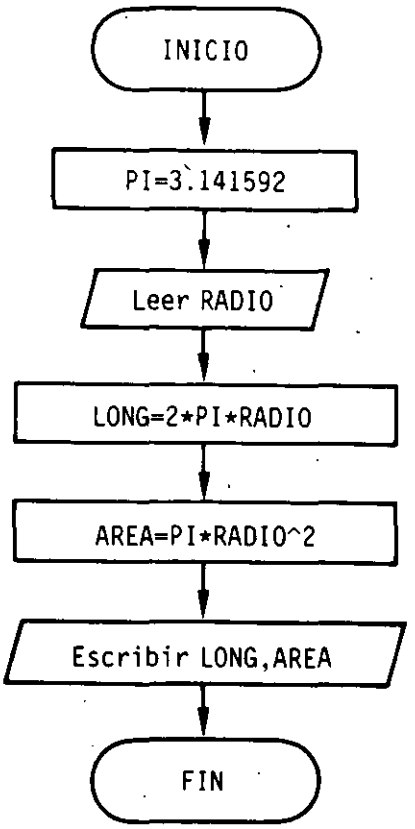
Existen normas I.S.O. (International Standard Organization), A.N.S.I. (American National Standard Institute), D.I.N., etc., y en España actualmente una prenorma dictada por el I.R.A.N.O.R. (Instituto de Racionalización y Normalización) que en un futuro próximo pasará a ser una norma U.N.E. (Una Norma Española).

Una herramienta común para el diseño de diagramas de flujo del sistema y del programa es la plantilla, consistente en una regla generalmente de plástico troquelada en la que aparecen los símbolos a utilizar, siendo su misión la de facilitar la representación de dichos diagramas.

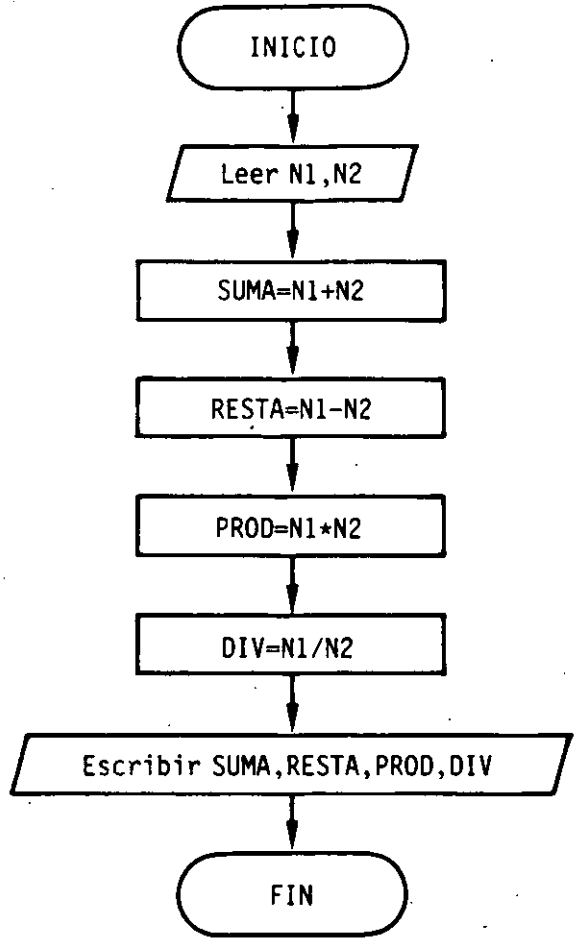


2.4. EJERCICIOS RESUELTOS

E.2.1. Programa que lee un número que corresponde al radio de una circunferencia y calcula e imprime la longitud de la misma y el área del círculo correspondiente.

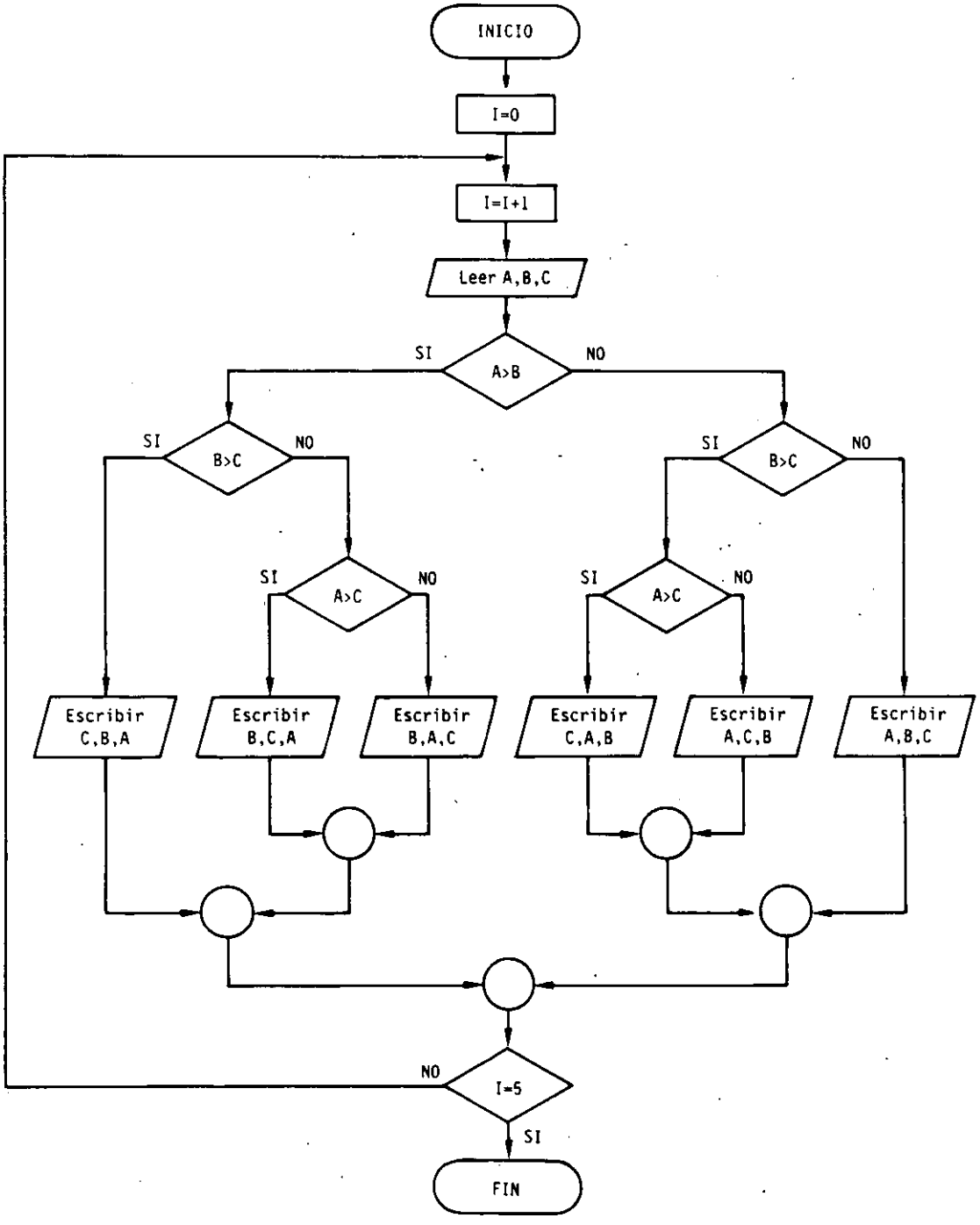


E.2.2. Programa que lee dos números y calcula e imprime su suma, resta, producto y división.



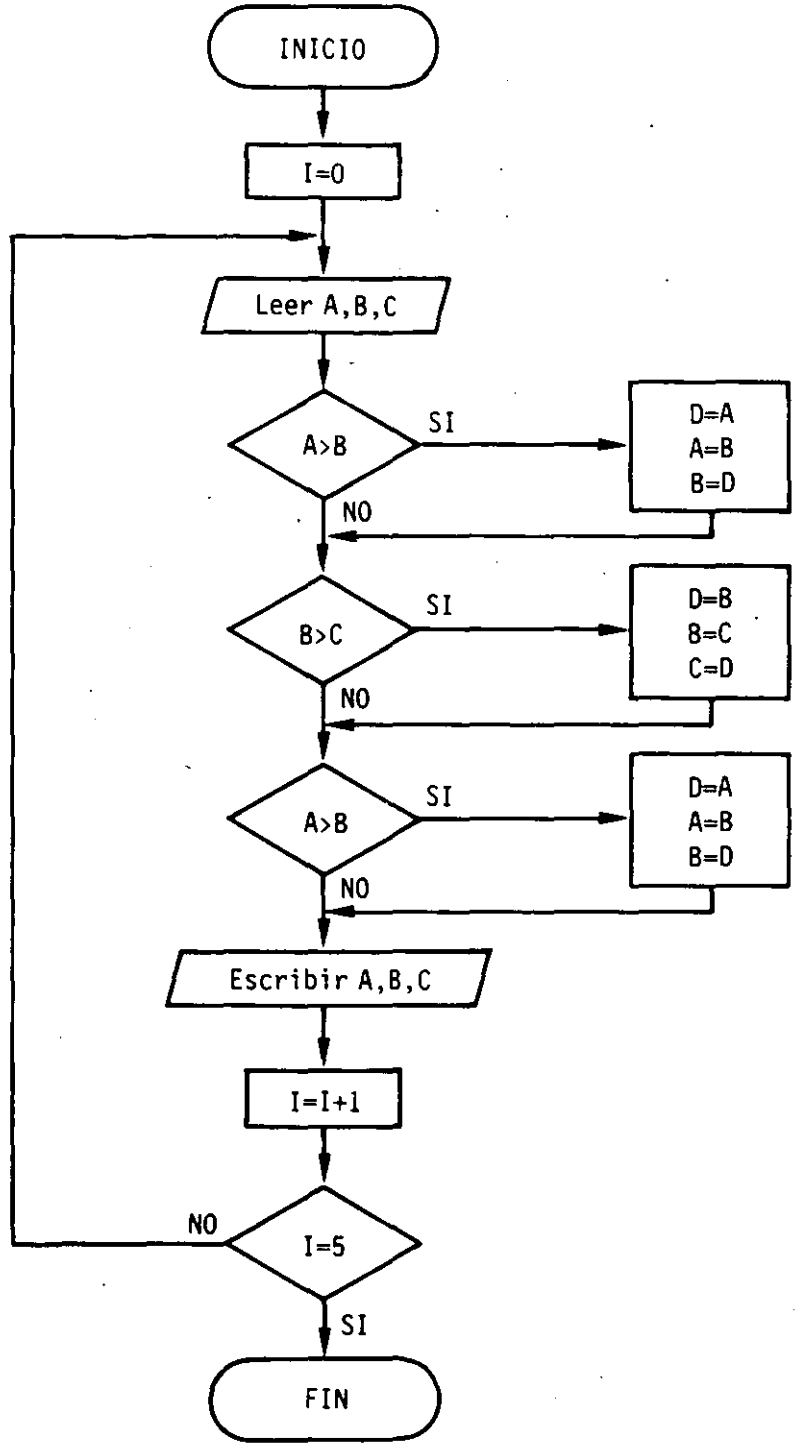
E.2.3. Programa que lea 5 veces 3 números y los imprima ordenados ascendentemente.

Los números se leen sobre 3 variables A, B, C, imprimiéndolas en el orden que corresponda.



E.2.4. Programa que lea 5 veces 3 números y los imprima ordenados ascendentemente.

Los números se leen sobre 3 variables A, B, C, imprimiéndolas en este mismo orden, siendo necesario por tanto el intercambio de sus contenidos.



Estructura general de un programa

3.0. INTRODUCCION

Un programa puede considerarse como una secuencia de acciones (instrucciones) que manipulan un conjunto de objetos (datos). Contendrá por tanto dos bloques para la descripción de los dos aspectos citados.

— Bloque de declaraciones: En él se especifican todos los objetos que utiliza el programa (constantes, variables, tablas, registros, archivos, etc.).
— Bloque de instrucciones: Constituido por el conjunto de operaciones que se han de realizar para la obtención de los resultados deseados.

En algunos lenguajes de programación no figura explícitamente el bloque de declaraciones (**FORTRAN, BASIC**), pero, en general, es necesario declarar todos los objetos que se van a usar (**COBOL, Pascal, PL/I, C, Ada**).

La ejecución de un programa consiste en la realización secuencial del conjunto de instrucciones, desde la primera a la última, de una en una. Este orden de realización únicamente será alterado mediante instrucciones denominadas de ruptura de secuencia.

Las instrucciones de un programa consisten, en general, en modificaciones sobre los objetos del programa, que constituyen su entorno, desde un estado inicial hasta otro final que contendrá los resultados del proceso.

El entorno de un programa u objetos del mismo, lo podemos considerar como el conjunto de recipientes necesarios para contener, tanto los datos de entrada, como los datos resultantes de todos los procesos que se lleven a cabo.

3.1. PARTES PRINCIPALES DE UN PROGRAMA

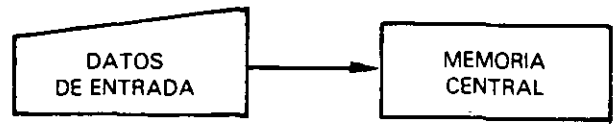
Dentro del bloque de instrucciones de un programa podemos diferenciar tres partes fundamentales.

En algunos casos, estas tres partes están perfectamente delimitadas; pero, en la mayoría, sus instrucciones quedan entremezcladas a lo largo del programa, si bien mantienen una cierta localización geométrica impuesta por la propia naturaleza de las mismas.

3.1.1. ENTRADA DE DATOS

La constituyen todas aquellas instrucciones que toman datos de un dispositivo externo, almacenándolos en la memoria central para que puedan ser procesados.

También se consideran dentro de este apartado las instrucciones de depuración de los datos de entrada, es decir, aquellas que se encargan de comprobar la corrección de los mismos.



3.1.2. PROCESO O ALGORITMO

Está formado por las instrucciones que modifican los objetos a partir de su estado inicial hasta el estado final, dejando éstos disponibles en la memoria central.

3.1.3. SALIDA DE RESULTADOS

Conjunto de instrucciones que toman los datos finales de la memoria central y los envían a los dispositivos externos.

3.2. CLASIFICACION DE LAS INSTRUCCIONES

Una instrucción se caracteriza por un estado inicial y final del entorno. El estado final de una instrucción coincide con el inicial de la siguiente.

No siempre una instrucción modifica el entorno, pues su cometido puede limitarse a una mera observación del mismo o a un cambio en el orden de ejecución de otras.

Según la función que desempeñan dentro del algoritmo, las instrucciones pueden ser:

3.2.1. INSTRUCCIONES DE DECLARACION

Se utilizan en aquellos lenguajes de programación que no tienen declaración explícita de los objetos. Su misión consiste en indicar al procesador que reserve espacio en la memoria para un objeto del programa, indicando asimismo su nombre, tipo y características.

Ejemplo:

En **BASIC** se declara un vector numérico V de 10 componentes mediante la instrucción.

```
DIM V(10)
```

3.2.2. INSTRUCCIONES PRIMITIVAS

Son aquellas que ejecuta el procesador de modo inmediato. Las principales son asignación, entrada y salida.

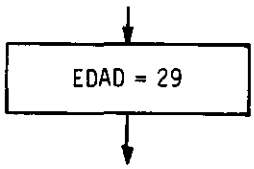
3.2.2.1. Instrucción de asignación

Consiste en calcular el valor de una expresión y almacenarlo en una variable. En algún lenguaje es preciso calcular previamente el resultado de la expresión, pues la instrucción de asignación sólo permite el movimiento de un valor simple.

Ejemplo:

Asignación del valor numérico entero 29 a la variable EDAD.

BASIC: EDAD = 29
COBOL: MOVE 29 TO EDAD
Pascal: EDAD := 29



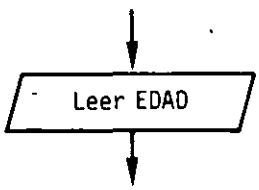
3.2.2.2. Instrucción de entrada

Toma un dato de un dispositivo de entrada y lo almacena en un objeto. En algún lenguaje, los datos de entrada no provienen de un dispositivo externo, sino que han sido colocados previamente en el mismo programa.

Ejemplo:

Entrada del dato numérico 29 a la variable EDAD.

BASIC: INPUT EDAD
READ EDAD
COBOL: ACCEPT EDAD
READ F-EDAD
Pascal: READ(EDAD)



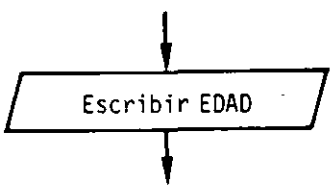
3.2.2.3. Instrucción de salida

Toma el valor de una expresión u objeto y lo lleva a un dispositivo externo.

Ejemplo:

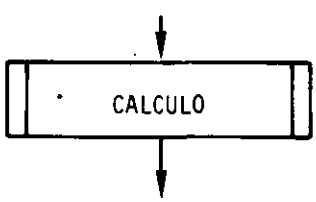
Salida del dato almacenado en la variable EDAD.

BASIC: PRINT EDAD
LPRINT EDAD
WRITE EDAD
COBOL: DISPLAY EDAD
WRITE EDAD
Pascal: WRITE(EDAD)



3.2.3. INSTRUCCIONES COMPUESTAS

Son aquellas que el procesador no puede ejecutar directamente, sino que realiza una llamada a un subprograma, subrutina o párrafo.



Ejemplo:

Instrucción compuesta consistente en la llamada a un bloque de instrucciones que realizan un **cálculo** determinado.

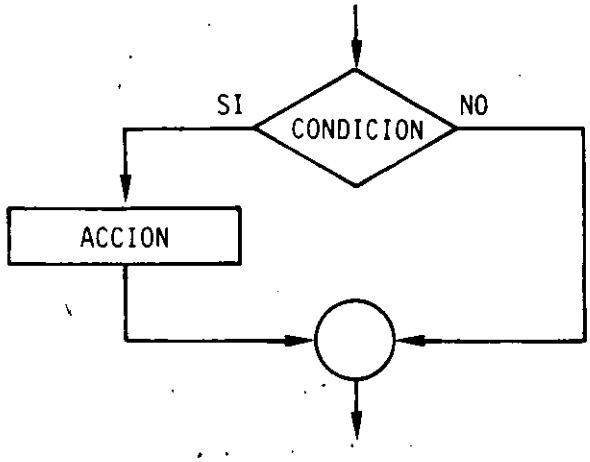
BASIC: GOSUB 1000
REM CALCULO es una subrutina que
comienza en la línea 1000
COBOL: PERFORM CALCULO
* CALCULO es un párrafo descrito aparte
Pascal: CALCULO
(*CALCULO es un procedimiento *)

3.2.4. INSTRUCCIONES DE CONTROL

Son aquellas encargadas de controlar la ejecución de otras instrucciones.

3.2.4.1. Instrucción alternativa

Controla la ejecución de unas u otras instrucciones según una condición.
Alternativa simple:

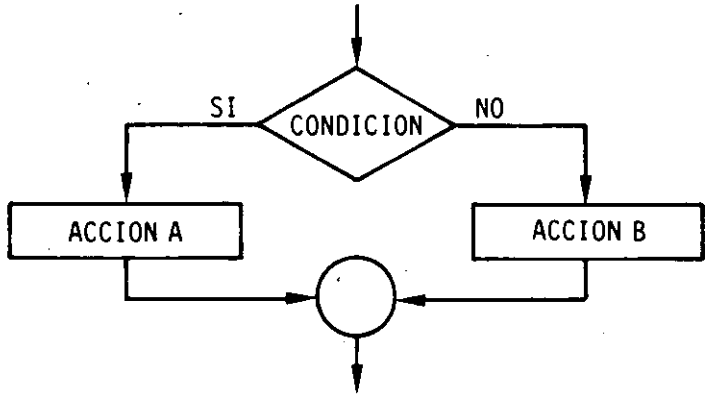


Ejemplo:

Instrucción que imprime el valor de A si se cumple que A > 0.

```
BASIC:    IF A > 0 THEN PRINT A
COBOL:    IF A > 0 DISPLAY A.
Pascal:   IF A > 0 THEN WRITE(A)
```

Alternativa doble:



Ejemplo:

Instrucción que imprime el valor de A si se cumple que A > B, y en caso contrario imprime B.

```
BASIC:    IF A > B THEN PRINT A
                   ELSE PRINT B
COBOL:    IF A > B DISPLAY A
             ELSE DISPLAY B.
Pascal:   IF A > B THEN WRITE(A)
                   ELSE WRITE(B)
```

3.2.4.2. Instrucción de salto incondicional

Altera la secuencia normal de ejecución de un programa, continuando la misma en la línea indicada en la propia instrucción.

Ejemplo:

Instrucción de salto incondicional a la línea de comienzo del programa.

```
BASIC:    GOTO 10
COBOL:    GO TO INICIO.
Pascal:   GOTO 1
```

3.2.4.3. Instrucción de salto condicional

Altera la secuencia normal de ejecución de un programa únicamente en el caso de cumplimiento de una condición asociada a la propia instrucción.

Ejemplo:

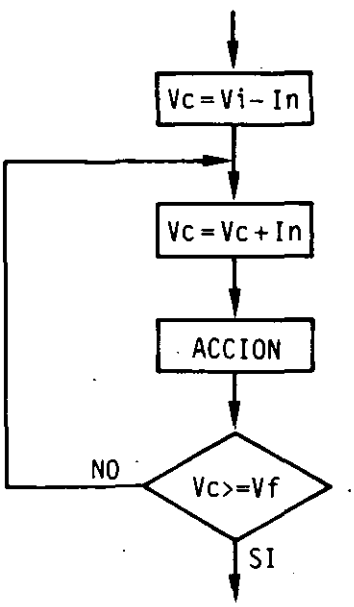
Instrucción de salto condicional a la línea de comienzo del programa si se cumple que A > 0.

```
BASIC:    IF A > 0 THEN GOTO 10
COBOL:    IF A > 0 GO TO INICIO.
Pascal:   IF A > 0 THEN GOTO 1
```

3.2.4.4. Instrucción repetitiva

Hace que se repitan una o varias instrucciones un número determinado o indeterminado de veces.

Instrucción PARA (FOR): El número de repeticiones está fijado de antemano.



Ejemplo:

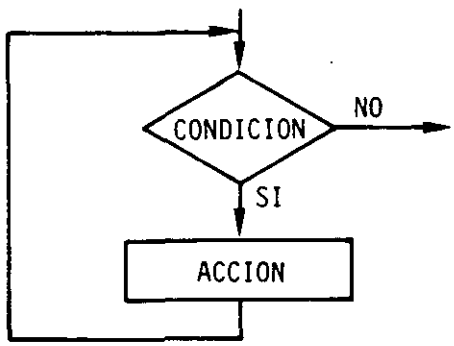
Escritura de los 10 primeros números naturales.

BASIC:
```
FOR N = 1 TO 10
PRINT N
NEXT N
```

COBOL:
```
PERFORM IMPRIMIR VARYING N FROM 1 BY 1 UNTIL N > 10.
IMPRIMIR.
    DISPLAY N.
```

Pascal:
```
FOR N := 1 TO 10 DO WRITE(N)
```

Instrucción MIENTRAS (WHILE): El número de repeticiones viene determinado por una condición que se ha de verificar antes de cada nueva repetición.



Ejemplo:

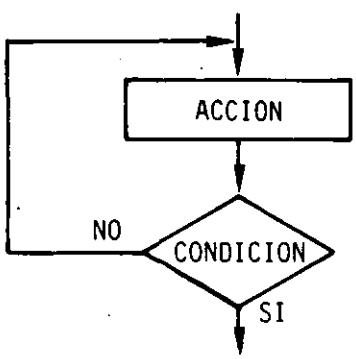
Lectura de N datos numéricos sobre un vector NUMEROS.

BASIC:
```
WHILE N > 0
INPUT NUMEROS(N)
N = N - 1
WEND
```

COBOL:
```
PERFORM LEER UNTIL N = 0.
LEER.
    ACCEPT NUMEROS(N).
    SUBTRACT 1 FROM N.
```

Pascal:
```
WHILE N > 0 DO
  BEGIN
    READ(NUMEROS[N]) ;
    N := N - 1
  END
```

Instrucción HASTA (UNTIL): La terminación del bucle depende del cumplimiento de una condición de salida que se evalúa después de cada repetición.



Ejemplo:

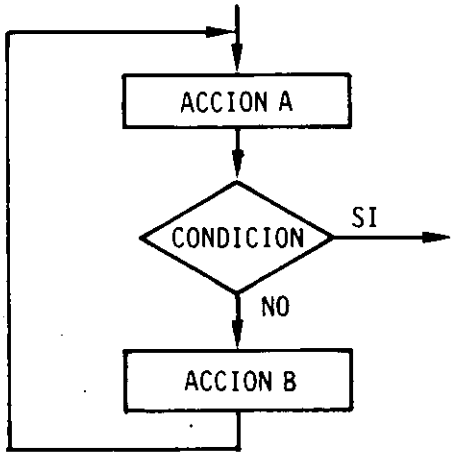
Lectura y escritura de nombres hasta que se lea el nombre "LUIS".

BASIC: No existe como instrucción en el lenguaje estándar.

COBOL:
```
PERFORM LECTURA-ESCRITURA UNTIL NOMBRE = 'LUIS'.
LECTURA-ESCRITURA.
    ACCEPT NOMBRE.
    DISPLAY NOMBRE.
```

Pascal:
```
REPEAT
  READ(NOMBRE);
  WRITELN(NOMBRE)
UNTIL NOMBRE = 'LUIS'
```

Instrucción ITERAR (LOOP): Se sale del bucle si se cumple una condición que se evalúa en cualquier lugar del mismo.



No existe en las versiones estándar de **BASIC, COBOL** y **Pascal.**

3.3. ELEMENTOS AUXILIARES DE UN PROGRAMA

Son variables que realizan funciones específicas dentro de un programa, y por su gran utilidad, frecuencia de uso y peculiaridades, conviene hacer un estudio separado de las mismas.

Las más importantes son los contadores, acumuladores e interruptores.

3.3.1. CONTADORES

Un contador es un campo de memoria cuyo valor se incrementa en una cantidad fija, positiva o negativa, generalmente asociado a un bucle.

Se utiliza en los siguientes casos:

— Para contabilizar el número de veces que es necesario repetir una acción (variable de control de un bucle).

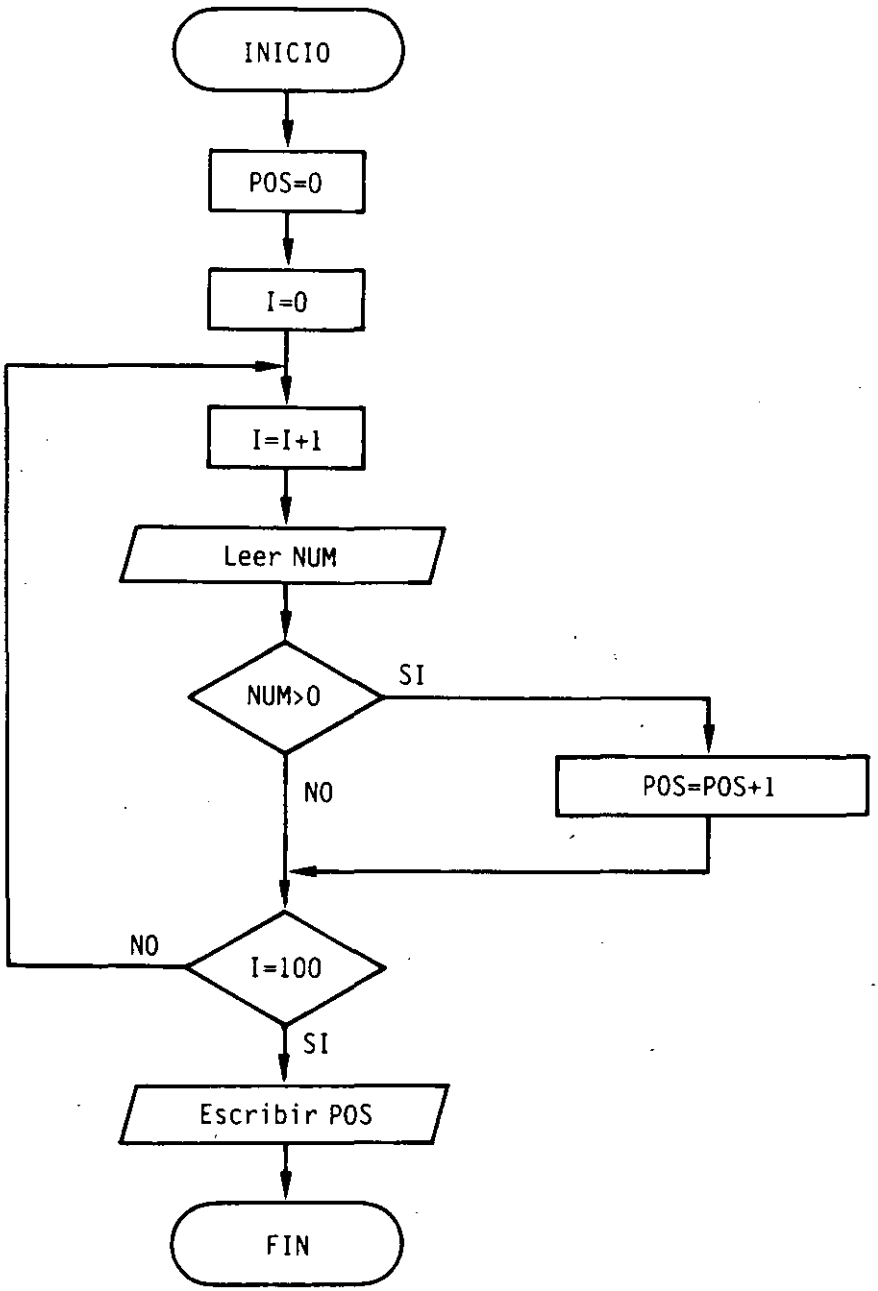
— Para contar un suceso particular solicitado por el enunciado del problema (asociado a un bucle o independientemente).

Un contador toma un valor inicial (0 en la mayoría de los casos) antes de comenzar su función, posteriormente, y cada vez que se realiza el suceso a contar, incrementa su valor (1 en la mayoría de los casos).

Ejemplo:

Programa que lea 100 números y cuente cuántos de ellos son positivos.

Se utilizan dos contadores, el primero I contabiliza la cantidad de números leídos, y el segundo POSITIVOS cuenta el resultado pedido.



3.3.2. ACUMULADORES

Un acumulador es un campo de memoria cuyo valor se incrementa sucesivas veces en cantidades variables.

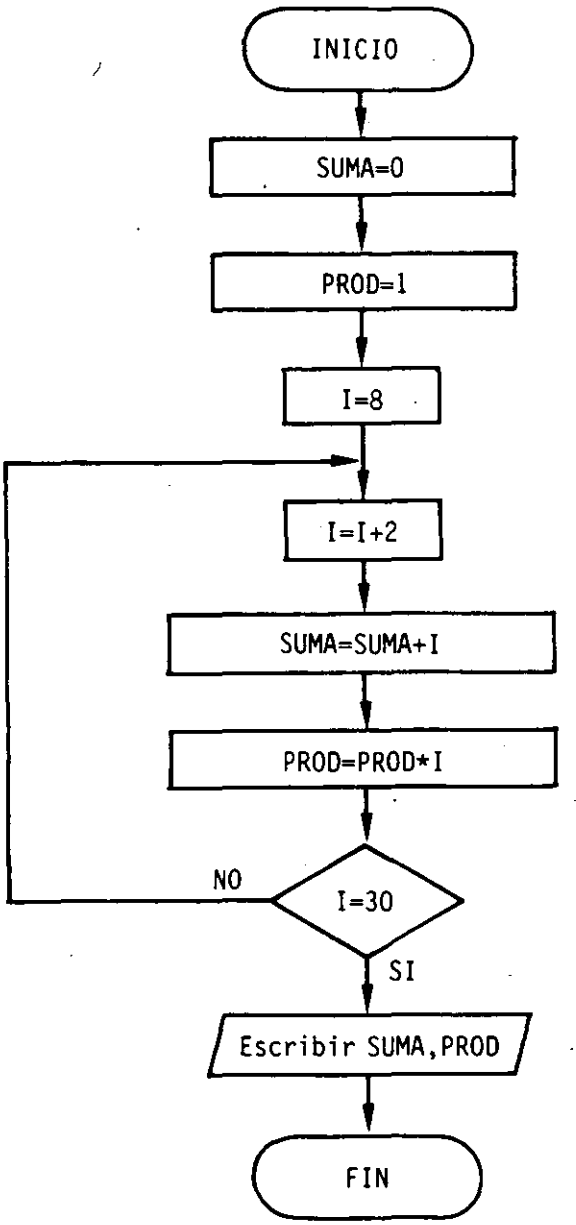
Se utiliza en aquellos casos en que se desea obtener el total acumulado de un conjunto de cantidades, siendo preciso inicializarlo con el valor 0.

También en las situaciones en que hay que obtener un total como producto de distintas cantidades se utilizá un acumulador, debiéndose inicializar con el valor 1.

Ejemplo:

Programa que calcule e imprima la suma y el producto de los números pares comprendidos entre el 10 y el 30 (ambos inclusive).

Se utilizan dos acumuladores SUMA y PRODUCTO que totalizan los dos resultados pedidos.



3.3.3. INTERRUPTORES (SWITCHES)

También se denominan *conmutadores*.

Un interruptor es un campo de memoria que puede tomar dos valores exclusivos (0 y 1, −1 y 1, FALSO y CIERTO, etc.).

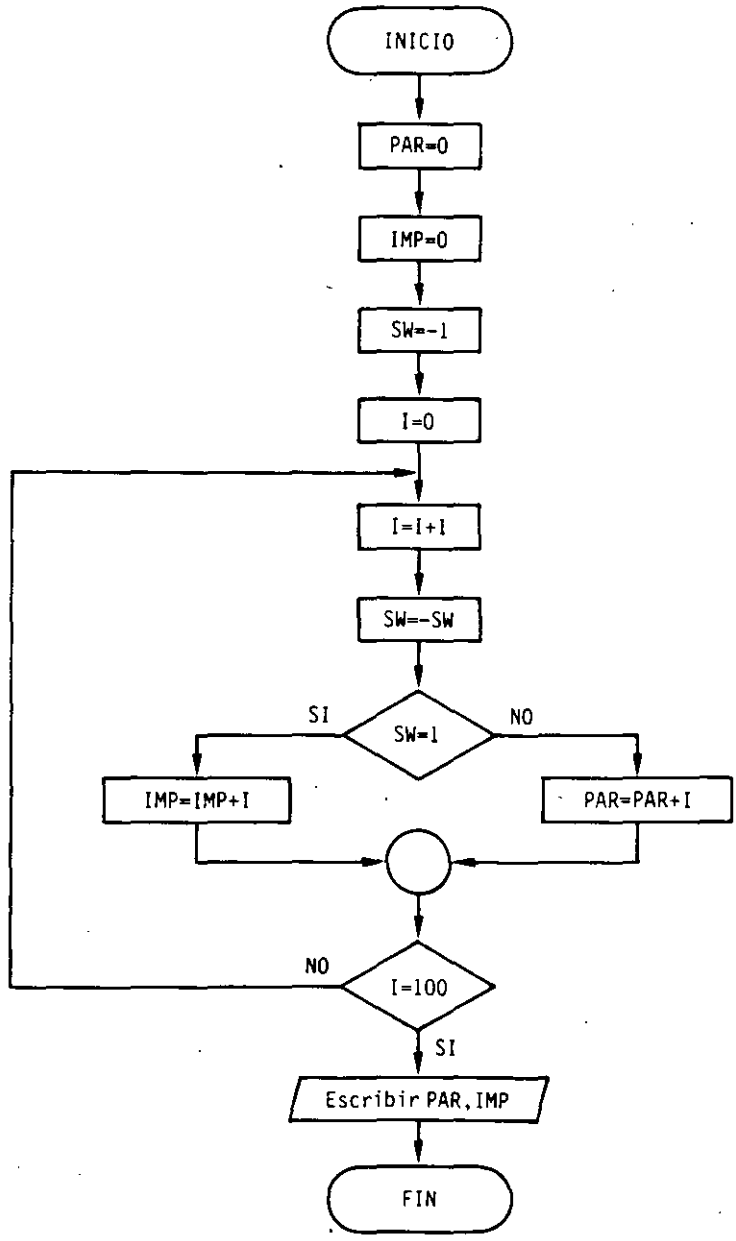
Se utiliza para:

— Recordar en un determinado punto de un programa la ocurrencia o no de un suceso anterior, para salir de un bucle o para decidir en una instrucción alternativa qué acción realizar.

— Para hacer que dos acciones diferentes se ejecuten alternativamente dentro de un bucle.

Ejemplo:

Programa que sume independientemente los pares y los impares de los números comprendidos entre 1 y 100.



3.4. TIPOS DE PROGRAMAS

Un programa, por lo general, estará compuesto por una secuencia de acciones, algunas de las cuales serán alternativas o repetitivas. En determinados programas sencillos, no se da esta mezcla de acciones, en cuyo caso los podemos clasificar como sigue:

— **Programas lineales.** Consisten en una secuencia de acciones primitivas (su ejecución es lineal en el orden en que han sido escritas).
— **Programas alternativos.** Consisten en el anidamiento de acciones alternativas (las tablas de decisión se realizan mediante programas alternativos).
— **Programas cíclicos.** Son aquellos en los que un conjunto de acciones se repiten un número determinado o indeterminado de veces (un programa de este tipo se denomina bucle).

Otra clasificación relativa a la aplicación desarrollada por el programa es:

— **Programas de gestión.** Se caracterizan por el manejo de gran cantidad de datos con pocos cálculos (resuelven problemas de gestión).
— **Programas técnico-científicos.** Al contrario que los anteriores, realizan gran cantidad de cálculos con pocos datos (resuelven problemas matemáticos, físicos, etc.).
— **Programas de diseño (CAD).** Se caracterizan por la utilización de técnicas gráficas para resolver problemas de diseño.
— **Programas de simulación.** Intentan reflejar una situación real, para facilitar su estudio.
— **Programas educativos (EAO).** Utilizan las ventajas del ordenador para la docencia.
— **Programas de inteligencia artificial.** Se utilizan para simular el razonamiento humano.
— Etcétera.

3.5. LENGUAJES DE PROGRAMACION

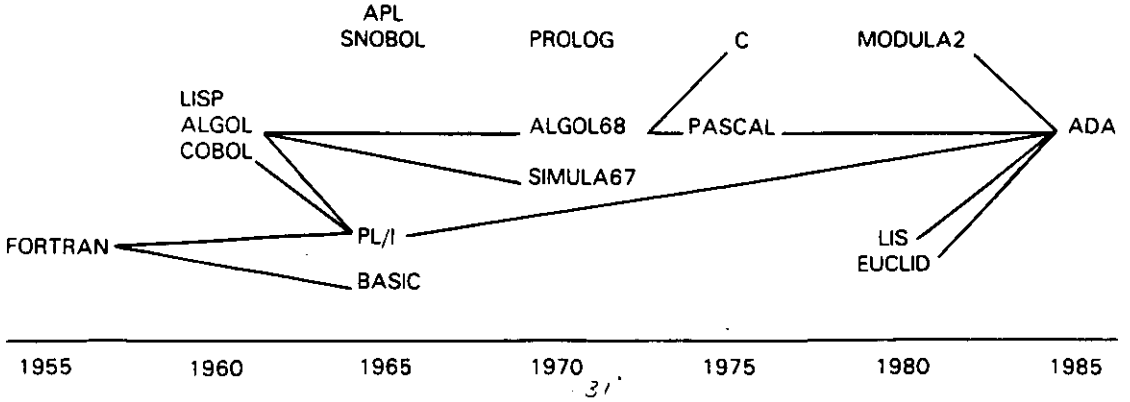
Un lenguaje de programación es una notación para escribir programas, es decir, para describir algoritmos dirigidos al computador.

Un lenguaje viene dado por una *gramática* o conjunto de reglas que se aplican a un *alfabeto*.

El primer lenguaje de programación que se utilizó fue el **lenguaje máquina,** el único que entiende directamente el computador, cuyo alfabeto es el binario, formado por los símbolos 0 y 1.

El **lenguaje ensamblador,** resultó de la evolución del lenguaje máquina, al sustituir las cadenas de símbolos binarios por nemotécnicos.

Posteriormente surgieron los **lenguajes de alto nivel,** cuya evolución (de los más importantes) es la del siguiente gráfico:



Los lenguajes de programación pueden clasificarse de la siguiente manera:

a) Según su parecido con el lenguaje natural.

— **Bajo nivel:** Lenguajes máquina y ensambladores.
— **Alto nivel:** Todos los demás.

b) Según la estructura de los programas.

— **Convencionales** o **línea a línea:** Ensambladores, FORTRAN, BASIC, COBOL, etc.
— **Estructurados:** Algol, PL/I, Pascal, Ada, COBOL estructurado, etc.

c) Según la realización de los programas.

— **Funcionales:** Lisp, Prolog, APL, etc.
— **Imperativos:** La mayoría.

d) Según el tipo de proceso.

— **Interactivos** o **conversacionales:** BASIC, Pascal, APL, etc.
— **Orientados al proceso por lotes (batch):** COBOL, FORTRAN, PL/I, etc.

Esta clasificación no es rigurosa puesto que algunos lenguajes pueden solaparse en uno u otro lugar.

3.6. EJERCICIOS RESUELTOS

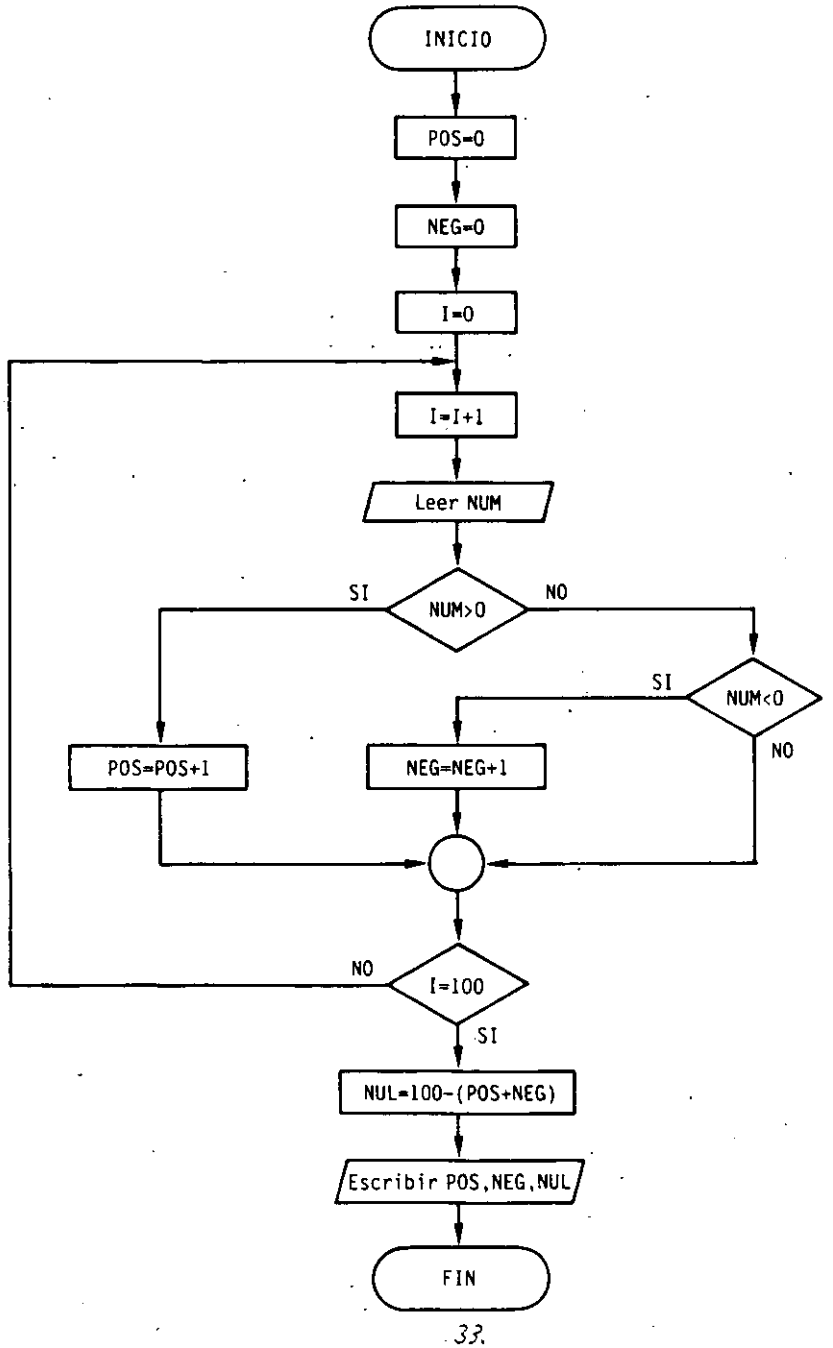
E.3.1. Programa que lea una secuencia de 100 números y obtenga e imprima cuántos hay positivos, negativos y nulos.

Se utilizan tres contadores:

— I para contar el número de lecturas.
— POSITIVOS para contar los mayores que 0.
— NEGATIVOS para contar los menores que 0.

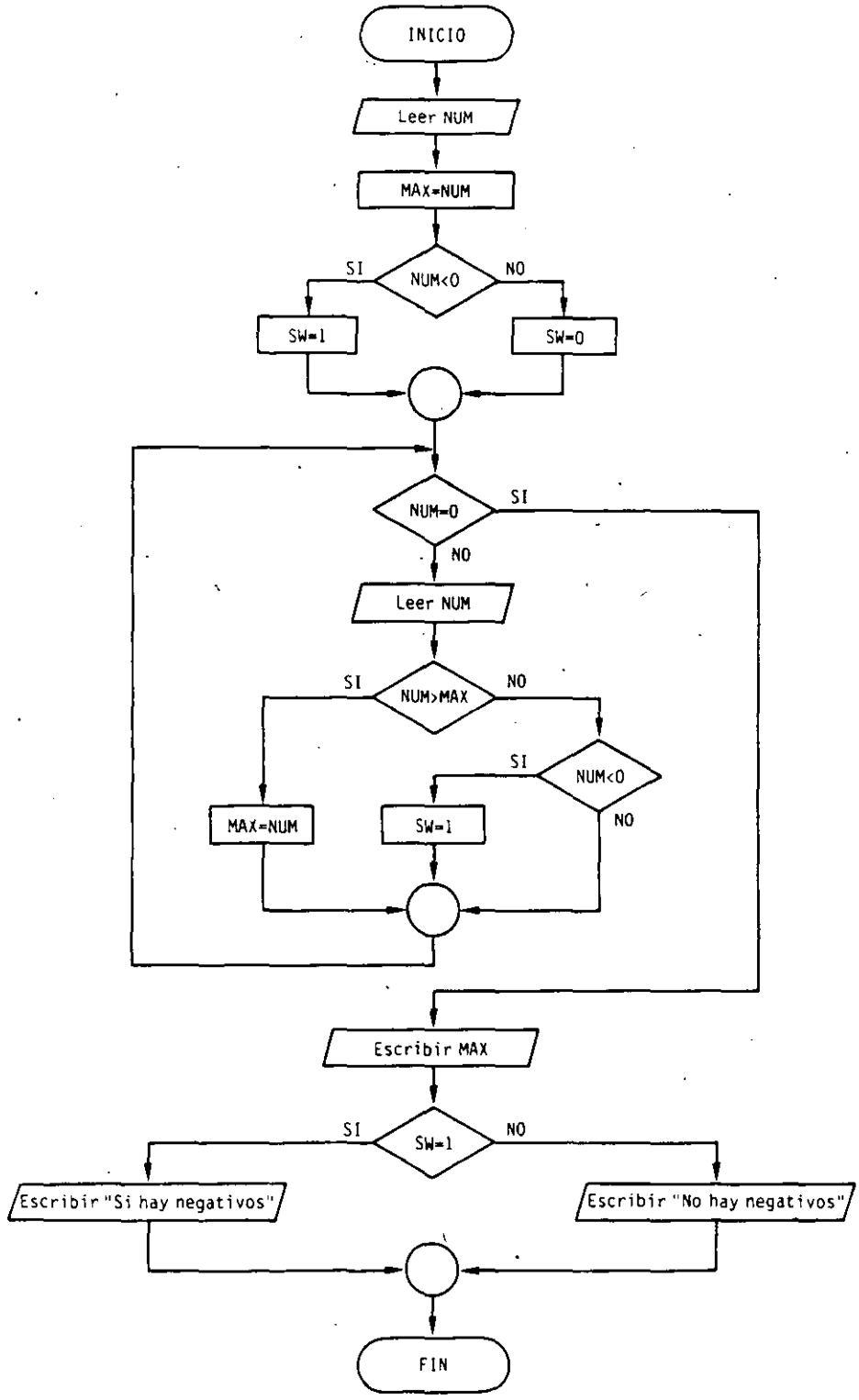
El número de NULOS no es necesario contarlo pues se calcula al final mediante la expresión:

NULOS = 100 - (POSITIVOS + NEGATIVOS)



E.3.2. Programa que lea una secuencia de números no nulos, terminada con la introducción de un 0, y obtenga e imprima el mayor, imprimiendo un mensaje de si se ha leído algún número negativo.

El programa consiste en un bucle de lectura, cuya condición de terminación es la lectura del número 0. Utiliza un interruptor que cambia de valor al detectarse el primer número negativo.



Notación pseudocodificada de programas

4.0. INTRODUCCION

La solución de un problema, dirigida a su ejecución por parte de un computador, requiere el uso de una notación que sea entendida por él, es decir, un lenguaje de programación. No obstante, durante la fase de diseño del programa, la utilización de un lenguaje así no es aconsejable debido a su rigidez.

Además de la utilización de las representaciones gráficas (ordinogramas), un programa puede describirse mediante un lenguaje intermedio entre el lenguaje natural y el lenguaje de programación, de tal manera que permita flexibilidad para expresar las acciones que se han de realizar y, sin embargo, imponga algunas limitaciones, importantes desde el punto de vista de su posterior codificación en un lenguaje de programación determinado.

Al igual que las otras técnicas, la utilización de una notación intermedia, permite el diseño del programa sin depender de ningún lenguaje de programación, y es posteriormente al diseño cuando se codificará el algoritmo obtenido en aquel lenguaje que nos interese.

4.1. PSEUDOCODIFICACION DE PROGRAMAS

Diremos que una notación es un **pseudocódigo** si mediante ella podemos describir la solución de un problema en forma de algoritmo dirigido al computador, utilizando palabras y frases del lenguaje natural sujetas a unas determinadas reglas.

El pseudocódigo se ha de considerar más bien una herramienta para el diseño de programas que una notación para la descripción de los mismos. Debido a su flexibilidad, permite obtener la solución a un problema mediante aproximaciones sucesivas, es decir, mediante lo que se denomina diseño descendente.

Todo pseudocódigo debe posibilitar la descripción de:

- Instrucciones de entrada/salida.
- Instrucciones de proceso.
- Sentencias de control del flujo de ejecución.
- Acciones compuestas, que hay que refinar posteriormente.

Asimismo, tendrá la posibilidad de describir, datos, tipos de datos, constantes, variables, expresiones, archivos y cualquier otro objeto que sea manipulado por el programa.

4.1.1. ACCIONES SIMPLES

Las acciones simples, también denominadas intrucciones primitivas, son aquellas que son ejecutadas de forma inmediata por el procesador.

4.1.1.1. Asignación

```
VARIABLE ← EXPRESION
```

Almacena en una variable el resultado de evaluar una expresión.

Ejemplo:

```
MEDIA ← SUMA / 6
```

4.1.1.2. Entrada

```
leer VARIABLE
```

Toma un dato del dispositivo estándar de entrada y lo almacena en una variable. Si se leen varias variables, se pueden colocar éstas en una misma instrucción separándolas por comas.

Ejemplo:

```
leer ALUMNO, CALIFICACION
```

4.1.1.3. Salida

```
escribir EXPRESION
```

Imprime en el dispositivo estándar de salida el resultado de evaluar una expresión. Al igual que en la lectura, se pueden imprimir varias expresiones en una sola instrucción de escritura.

Ejemplo:

```
escribir SUMAMEDIAS / NUMALUMNOS
```

Las instrucciones leer y escribir no son en realidad acciones primitivas en ningún lenguaje de programación, pero conviene considerarlas así desde el punto de vista del diseño del programa.

4.1.2. SENTENCIAS DE CONTROL

También se denominan sentencias estructuradas y controlan el flujo de ejecución de otras instrucciones.

4.1.2.1. Secuencia

```
I1 ; I2 ; I3 ; ... In
```

```
I1
I2
...
In
```

Se ejecutan las instrucciones I1, I2, ... In, en el mismo orden en que aparecen escritas. Utilizamos el punto y coma como separador de instrucciones que están en la misma línea.

Ejemplo:

```
leer NOTA
SUMA ← SUMA + NOTA
MEDIA ← SUMA / 6
escribir MEDIA
```

4.1.2.2. Alternativa

- Alternativa doble:

```
si CONDICION
   entonces I1 ; I2 ; ... In
   sino     J1 ; J2 ; ... Jk
finsi
```

En esta instrucción la condición es una expresión booleana, si su evaluación produce el resultado CIERTO se ejecutarán las instrucciones I1, I2, ... In, y en caso contrario las J1, J2, ... Jk.

Ejemplo:

```
si NOTA < 5
   entonces escribir "SUSPENSO"
   sino     escribir "APROBADO"
finsi
```

- Alternativa simple:

Un caso particular de la instrucción anterior, es en el que no hay que ejecutar ninguna instrucción si la condición produce resultado FALSO.

```
si CONDICION
   entonces I1 ; I2 ; ... In
finsi
```

Ejemplo:

```
si NOTA > 0
   entonces SUMA ← SUMA + NOTA
finsi
```

4.1.2.3. Repeticiones o bucles

En todo bucle hay una o varias acciones que se han de repetir y una condición que determina el número de repeticiones de las mismas. Es fundamental que el valor de la condición sea afectado por las acciones para asegurar la terminación del bucle en algún momento.

Según si la evaluación de la condición se realiza al comienzo, al final o dentro del bucle, se tienen las siguientes sentencias:

- **Mientras (While):**

```
mientras CONDICION hacer
   I1 ; I2 ; ... In
finmientras
```

Se evalúa la condición antes de iniciar el bucle, y se repiten sucesivamente las instrucciones I1, I2, ... In, mientras siga siendo CIERTA.

Ejemplo:

```
mientras IN = 0 hacer
   leer NOTA
   si NOTA > 0
      entonces SUMA ← SUMA + NOTA
      sino     IN ← 1
   finsi
finmientras
```

- **Hasta (Until):**

```
repetir
   I1 ; I2 ; ... In
hasta CONDICION
```

Se evalúa la condición después de cada ejecución de las instrucciones I1, I2, ... In, y se termina el bucle si es CIERTA.

Ejemplo:

```
repetir
   leer NOTA
   SUMA ← SUMA + NOTA
hasta NOTA = 0
```

- **Para (For):**

```
para Vc de Vi a Vf con incremento In hacer
   I1 ; I2 ; ... In
finpara
```

Se repiten las instrucciones I1, I2, ... In, un número fijo de veces, tantas como sucesivos valores toma la variable de control del bucle Vc desde, inicialmente Vi, incrementándose a cada repetición en In, hasta que el valor de Vc supera Vf.

Si el incremento es +1, que es el caso más usual, el bucle se expresa:

```
para Vc de Vi a Vf hacer
   I1 ; I2 ; ... In
finpara.
```

Ejemplo:

```
Para I de 1 a 6 hacer
   leer NOTA
   SUMA ← SUMA + NOTA
finpara
```

- Iterar (**Loop**):

```
iterar
   I1 ; I2; ... In
   salir si CONDICION
   J1 ; J2 ; ... Jk
finiterar
```

Se ejecutan las instrucciones I1, I2, ... In, a continuación se evalúa la condición de salida del bucle y si no es CIERTA se ejecutan J1, J2, ... Jk, repitiéndose de nuevo el proceso hasta que la condición sea CIERTA.

Ejemplo:

```
iterar
   leer NOMBRE
   salir si NOMBRE = "FIN"
   leer NOTA1, NOTA2
   MEDIA ← (NOTA1 + NOTA2) / 2
   escribir NOMBRE, MEDIA
finiterar
```

Los bucles **mientras, hasta** y **para** son casos particulares del anterior, es decir, siempre se puede utilizar un bucle **iterar** en lugar de cualquiera de los otros, aunque cada uno de ellos se adapta mejor a una determinada situación.

4.1.3. ACCIONES COMPUESTAS

Una acción compuesta es aquella que ha de ser realizada dentro del algoritmo, pero que aún no está resuelta en términos de acciones simples y sentencias de control.

En el diseño del programa se incluirán los nombres de las acciones compuestas en el algoritmo y posteriormente habrá que refinarlas, sustituyendo cada nombre por las instrucciones correspondientes o colocándolas aparte, mediante lo que se denomina subprograma, como veremos más adelante en el Capítulo 10.

4.1.4. COMENTARIOS

Son líneas explicativas cuyo objetivo es facilitar la comprensión del programa a quien lo lea. Estas líneas serán ignoradas por el procesador cuando ejecute el programa.

Un programa, en cualquier lenguaje, debe estar ampliamente documentado mediante comentarios intercalados a lo largo de todo su listado. Esto facilitará las posibles y necesarias modificaciones del mismo al simplificar su comprensión al programador que lo diseñó o a otro diferente encargado de su mantenimiento.

Los comentarios se utilizan para aclarar:

— El significado o cometido de un objeto del programa.

— El objetivo de un bloque de instrucciones.
— La utilización de una determinada instrucción.
— Siempre que sea necesario aclarar algún aspecto del programa.

En la fase de diseño del programa no es necesario excederse en ellos, pero sí se han de incluir aquellos que consideremos necesarios.

Se escribirán en cualquier línea a continuación de los símbolos **.

```
** Comentario de aclaración
```

4.1.5. OBJETOS DEL PROGRAMA

Podemos considerarlos como los recipientes de los datos que manipula el programa. Será necesario indicar cuáles son sus nombres y sus tipos, lo que se hará previamente a la descripción del conjunto de instrucciones que forman el algoritmo.

Al conjunto de objetos de un programa se denomina **entorno.**

Ejemplo:

```
Entorno:
  I es numérica entera
  NOTA es numérica real
  NOMBRE es alfanumérica
```

En la descripción anterior se declaran 3 variables, I que admitirá valores numéricos enteros, NOTA reales y NOMBRE cadenas de caracteres.

4.1.6. PROGRAMA

Un programa es la solución final de un problema. En esta notación consiste en la descripción de los objetos **(entorno)** y de las instrucciones **(algoritmo).**

Tendrá una cabecera con el nombre del programa y dos bloques precedidos por las palabras "entorno:" y "algoritmo:".

```
Programa NOMBRE DEL PROGRAMA
Entorno:
  ** descripción de los objetos
  ...
Algoritmo:
  ** descripción de las acciones
  ...
Finprograma
```

Ejemplo:

Generación de actas. Se introducen por teclado una secuencia de informaciones, cada una de ellas compuesta por un nombre y 6 números, correspondientes al nombre de un alumno y las calificaciones que ha obtenido en sus 6 asignaturas. La secuencia termina al introducir el nombre "FIN".

Se desea un programa que imprima un listado de calificaciones, en el que ha de figurar el nombre del alumno seguido de su nota media. Finalmente se imprimirá la nota media del grupo.

La entrada de datos será de la forma:

```
EMILIO PEREZ GARCIA
5, 8, 7, 5, 6, 6
...
ANA CASAS ORTIZ
5, 3, 6, 2, 7, 4
FIN
```

El listado proporcionado será:

```
        LISTADO DE CALIFICACIONES
        -------------------------
NOMBRE DEL ALUMNO          NOTA MEDIA
-----------------          ----------
EMILIO PEREZ GARCIA           6.1
...                           ...
ANA CASAS ORTIZ               4.5

      NOTA MEDIA DEL GRUPO: 5.7
```

Diseño del programa.

```
Algoritmo:
  imprimir cabeceras del listado
  inicializar acumuladores
  iterar
    leer NOMBRE
    salir si NOMBRE = "FIN"
    contabilizar alumno
    leer notas del alumno
    calcular su nota media
    acumular nota media
    imprimir línea con alumno y nota media
  finiterar
  obtener nota media del grupo
  imprimir nota media del grupo
Finalgoritmo
```

Mediante refinamientos del algoritmo anterior, obtenemos el siguiente programa:

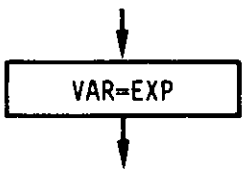
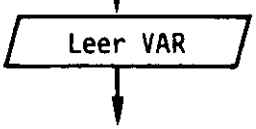
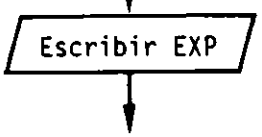
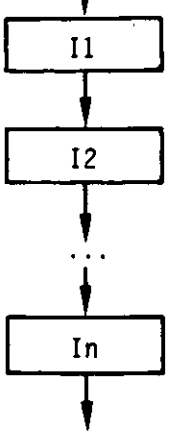
```
Programa GENERACION DE ACTAS
Entorno:
  NOMBRE es alfanumérica
  I, NUMALUMNOS son numéricas enteras
  NOTA, SUMA, MEDIA, SUMAMEDIA, MEDIAGRUPO son numéricas reales
Algoritmo:
  escribir "      LISTADO DE CALIFICACIONES"
  escribir "      -------------------------"
  escribir "NOMBRE DEL ALUMNO        NOTA MEDIA"
  escribir "-----------------        ----------"
  NUMALUMNOS ← 0
  SUMAMEDIA ← 0
  iterar
  leer NOMBRE
  salir si NOMBRE = "FIN"
  NUMALUMNOS ← NUMALUMNOS + 1
  SUMA ← 0
```

```
        para I de 1 a 6 hacer
          leer NOTA
          SUMA ← SUMA + NOTA
        finpara
        MEDIA ← SUMA / 6
        SUMAMEDIA ← SUMAMEDIA + MEDIA
        escribir NOMBRE, MEDIA
      finiterar
      MEDIAGRUPO ← SUMAMEDIA / NUMALUMNOS
      escribir " NOTA MEDIA DEL GRUPO:", MEDIAGRUPO
    Finprograma
```

4.2. PASO DE PSEUDOCODIGO A DIAGRAMA DE FLUJO

La traducción es totalmente mecánica y no presenta ningún problema. La traducción inversa, de diagrama de flujo a pseudocódigo, no siempre es posible.

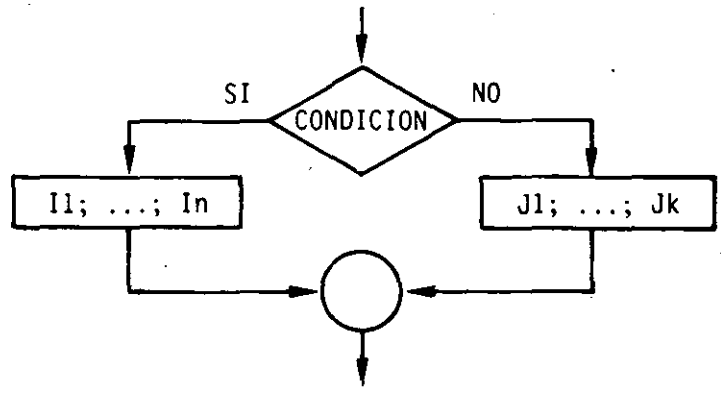
A continuación figuran las estructuras equivalentes en ambas notaciones.

PSEUDOCODIGO	ORDINOGRAMA
VARIABLE ← EXPRESION	
leer VARIABLE	
escribir EXPRESION	
I1; I2; ... In	

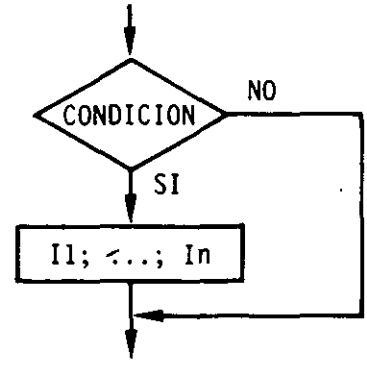
PSEUDOCODIGO

```
si CONDICION
  entonces I1; ... In
  sino     J1; ... Jk
finsi
```

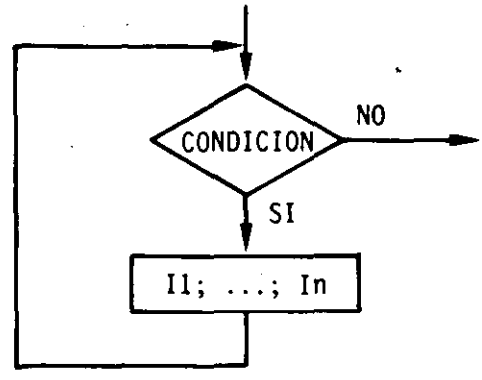
ORDINOGRAMA



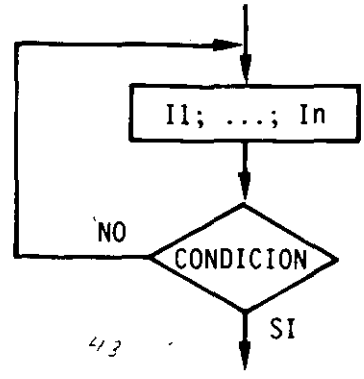

```
si CONDICION
  entonces I1; ... In
finsi
```




```
mientras CONDICION hacer
  I1; ... In
finmientras
```



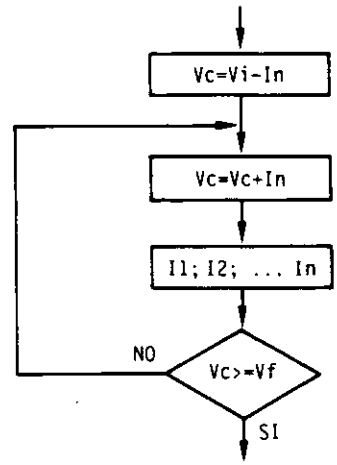

```
repetir
  I1; ... In
hasta CONDICION
```



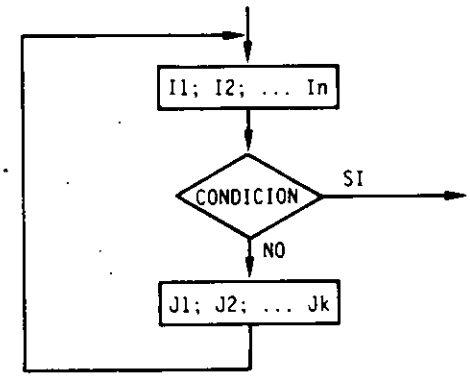
PSEUDOCODIGO

ORDINOGRAMA

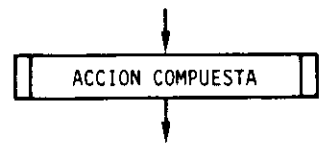
```
para Vc de Vi a Vf
  con incremento In hacer
  I1; ... In
finpara
```




```
iterar
  I1; ... In
  salir si CONDICION
  J1; ... Jk
finiterar
```

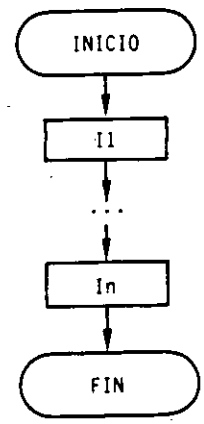



```
ACCION COMPUESTA
```




```
** Comentario
```

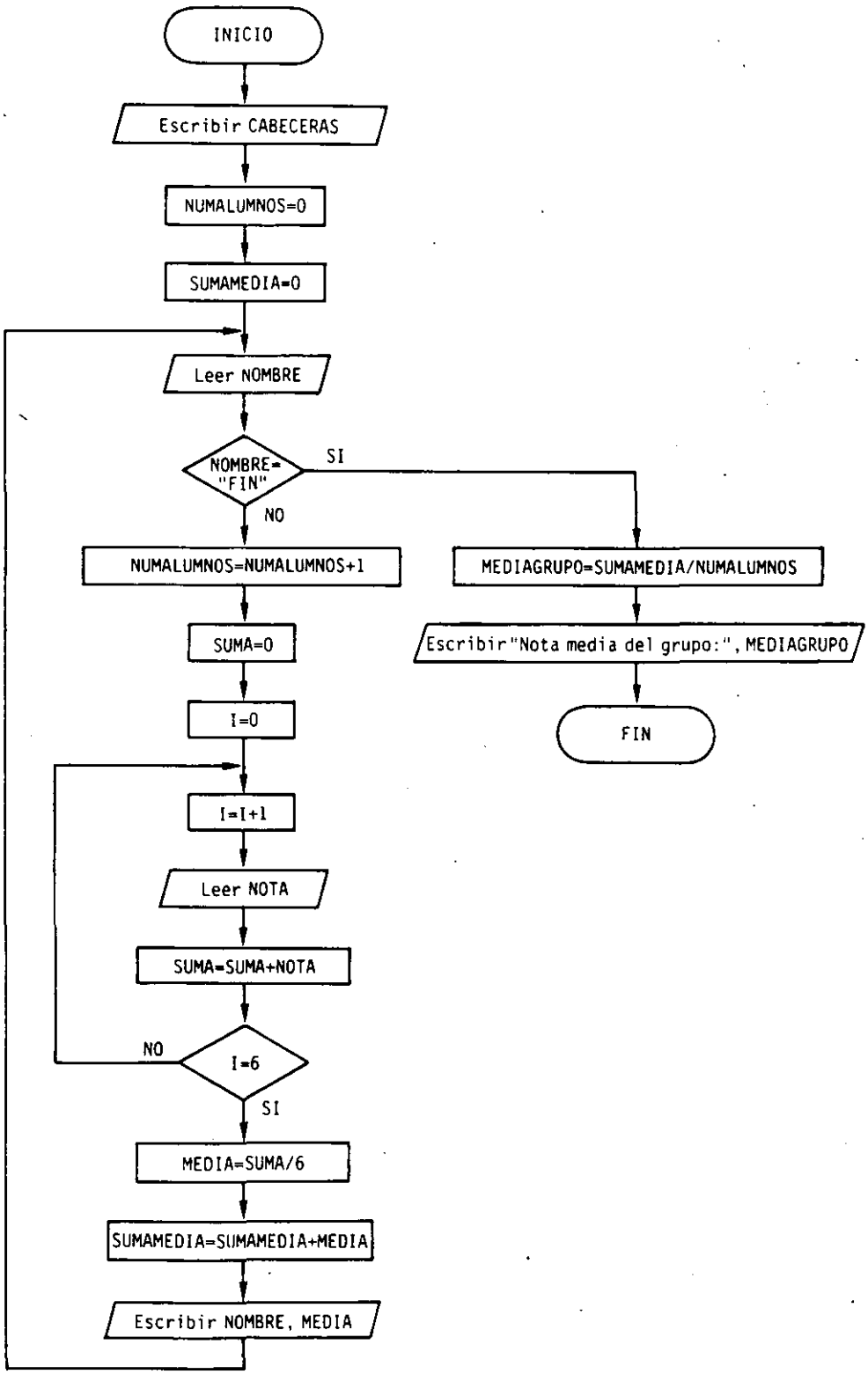

```
Algoritmo:
  I1
  ...
  In
Finprograma
```



Ejemplo:

Ordinograma correspondiente al programa anterior.

GENERACION DE ACTAS



4.3. PASO DE PSEUDOCODIGO A LENGUAJE DE PROGRAMACION

La codificación en un lenguaje de programación de un programa escrito mediante pseudocódigo es, al igual que el paso a ordinograma, bastante sencilla.

Algunas sentencias no tienen explícitamente una instrucción equivalente en algún lenguaje, por lo cual será preciso implementarla mediante los recursos disponibles en dicho lenguaje.

A continuación se da una traducción de las sentencias del pseudocódigo a los lenguajes **BASIC, COBOL** y **Pascal**.

Para ampliar el conocimiento de cada uno de los lenguajes se recomienda leer el manual correspondiente o algunos de los libros referenciados en la bibliografía.

- Asignación:

```
VARIABLE ← EXPRESION
```

BASIC: `LET VARIABLE = EXPRESION`

La palabra reservada LET es opcional y no suele usarse.

COBOL: `COMPUTE VARIABLE = EXPRESION.`

Para asignar un valor o el contenido de un campo se utiliza también la siguiente instrucción:

```
MOVE VALOR TO VARIABLE.
```

Existen además instrucciones particulares para los casos concretos en que la expresión es una suma (ADD), resta (SUBTRACT), producto (MULTIPLY) o división (DIVIDE).

Pascal: `VARIABLE := EXPRESION`

- Entrada:

```
Leer VARIABLE
```

BASIC: `INPUT VARIABLE` *(Entrada por teclado)*
`READ VARIABLE` *(Lectura de líneas DATA)*
COBOL: `ACCEPT VARIABLE.`
Pascal: `READ(VARIABLE)`

- Salida:

```
Escribir EXPRESION
```

BASIC: `PRINT EXPRESION` *(Escritura en pantalla)*
`LPRINT EXPRESION` *(Escritura en impresora)*
`WRITE EXPRESION` *(Escritura en pantalla)*
COBOL: `DISPLAY DATO.`

Si se quiere escribir el resultado de una expresión que implique operaciones, es necesario realizarlas previamente con una sentencia de asignación y a continuación escribir el resultado.

Pascal: `WRITE(EXPRESION)`

- Secuencia:

```
I1; I2; ... In
```

BASIC: La secuencia viene dada por los números de línea en orden creciente.

```
250 I1
260 I2        o también        250 I1: I2: ... In
...
500 In
```

COBOL: La secuencia viene dada por el orden de escritura de las instrucciones, respetando las reglas de formato del lenguaje.

```
I1.
I2.        o también        I1 I2 ... In.
...
In.
```

Pascal: Las instrucciones se escriben con formato libre, separándolas con punto y coma.

```
I1;
I2;        o también        I1; I2; ... In
...
In
```

- Alternativa simple:

```
si CONDICION
  entonces I1; ... In
finsi
```

BASIC:

```
IF CONDICION THEN I1: ... In
```

COBOL:

```
IF CONDICION I1 ... In.
```

Pascal:

```
IF CONDICION
    THEN BEGIN
           I1; ... In
         END
```

- Alternativa doble:

```
si CONDICION
  entonces I1; ... In
  sino     J1; ... Jk
finsi
```

BASIC:

```
IF CONDICION THEN I1: ... In ELSE J1: ... Jk
```

COBOL:
```
IF CONDICION I1 ... In
      ELSE  J1 ... Jk.
```

Pascal:
```
IF CONDICION
  THEN BEGIN
         I1; ... In
       END
  ELSE BEGIN
         J1; ... Jk
       END
```

- Bucle mientras:

```
mientras CONDICION hacer
  I1; ... In
finmientras
```

BASIC:
```
250 WHILE CONDICION
260 I1
...
490 In
500 WEND
```

COBOL:
```
PERFORM SECUENCIA UNTIL CONDICION.
```

SECUENCIA es un párrafo que figura en otro lugar del programa y que contiene las sentencias I1, ... In.

Pascal:
```
WHILE CONDICION DO
        BEGIN
          I1; ... In
        END
```

- Bucle hasta:

```
repetir
  I1; ... In
hasta CONDICION
```

BASIC: Se implementa mediante un salto condicional.

```
250 I1
...
490 In
500 IF NOT CONDICION THEN GOTO 250
```

COBOL: No existe explícitamente esta sentencia, se implementa de la siguiente manera:

```
BUCLE.
  I1 ... In.
  IF NOT CONDICION GO. TO BUCLE.
```

Pascal:
```
REPEAT
  I1; ... In
UNTIL CONDICION
```

- Bucle para:

```
para Vc de Vi a Vf con incremento In hacer
  I1; ... In
finpara
```

BASIC:
```
250 FOR Vc = Vi TO Vf STEP In
260 I1
...
490 In
500 NEXT Vc
```

No es necesario escribir "STEP In" si In = + 1.

COBOL:
```
PERFORM S VARYING Vc FROM Vi BY In UNTIL Vc > Vf.
```

S es un párrafo que figura en otro lugar del programa y que contiene las sentencias I1, ... In.

Pascal: Tiene dos sentencias para incrementos +1 y −1, para incrementos diferentes es necesario implementarlo a partir de un bucle WHILE.

```
FOR Vc := Vi TO Vf DO (* incremento +1 *)
  BEGIN
    I1; ... In
  END

FOR Vc := Vi DOWNTO Vf DO (* incremento -1 *)
  BEGIN
    I1; ... In
  END
```

- Bucle iterar:

```
iterar
  I1; ... In
  salir si CONDICION
  J1; ... Jk
finiterar
```

No existe explícitamente en ninguno de los tres lenguajes, por lo que se implementará como sigue:

BASIC:
```
250 I1
...
340 In
350 IF CONDICION THEN GOTO 500
360 J1
...
480 Jk
490 GOTO 250
500 ...
```

COBOL:
```
BUCLE.
    I1 ... In.
    IF CONDICION GO TO FIN-BUCLE.
    J1 ... Jk.
    GO TO BUCLE.
FIN-BUCLE.
    ...
```

Pascal:
```
I1; ... In;
WHILE NOT CONDICION DO
  BEGIN
    J1; ... Jk;
    I1; ... In
  END
```

- Comentarios:

```
** comentario
```

BASIC: Ocupa una línea que empieza con la palabra reservada REM.

```
350 REM comentario
```

COBOL: Ocupa una línea en la cual se ha escrito un "*" en la columna 7.

```
*     comentario
```

Pascal: Se coloca en cualquier lugar, encerrado entre "(*" y "*)".

```
(* comentario *)
```

- Programa:

```
Programa NOMBRE DEL PROGRAMA
...
Finprograma
```

BASIC: Está formado por una serie de líneas numeradas en orden creciente comenzando con el nombre de programa como comentario y terminando con la sentencia END.

```
10 REM NOMBRE DEL PROGRAMA
...
50 END
```

COBOL: Su forma puede variar según los casos, siendo la más generalizada.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. NOMBRE-DEL-PROGRAMA.
...
    STOP RUN.
...
```

Pascal: Consta de un encabezamiento con el nombre del programa y los archivos utilizados, un bloque de declaraciones (etiquetas, constantes, tipos, variables y subprogramas) y un bloque de instrucciones entre "BEGIN" y "END.".

```
PROGRAM NOMBREDELPROGRAMA (archivos);
...
BEGIN (*PP*)
  ...
END. (*PP*)
```

Ejemplo:

Codificación del programa GENERACION DE ACTAS.

Codificación BASIC:

```
10 REM GENERACION DE ACTAS
20 CLS
30 LPRINT "      LISTADO DE CALIFICACIONES"
40 LPRINT "      ------------------------": LPRINT
50 LPRINT "NOMBRE DEL ALUMNO      NOTA MEDIA"
60 LPRINT "----------------       ----------"
70 NUMALUMNOS = 0
80 SUMAMEDIA = 0
90 INPUT "Escriba nombre del alumno o FIN "; NOMBRE$
100 IF NOMBRE$ = "FIN" THEN GOTO 220
110 NUMALUMNOS = NUMALUMNOS + 1
120 SUMA = 0
130 FOR I = 1 TO 6
140   PRINT "Escriba nota número "; I;
150   INPUT NOTA
160   SUMA = SUMA + NOTA
170 NEXT I
180 MEDIA = SUMA / 6
190 SUMAMEDIA = SUMAMEDIA + MEDIA
200 LPRINT NOMBRE$, MEDIA
210 GOTO 90
220 REM Cálculo de la nota media del grupo
230 MEDIAGRUPO = SUMAMEDIA / NUMALUMNOS
240 LPRINT
250 LPRINT "   NOTA MEDIA DEL GRUPO: "; MEDIAGRUPO
260 END
```

Codificación COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. GENERACION-DE-ACTAS.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  NOMBRE PIC X(25).
77  I      PIC 9.
77  NUMALUMNOS PIC 999.
77  NOTA   PIC 99V99.
```

```
77  SUMA    PIC 99V99.
77  MEDIA   PIC 99V99.
77  SUMAMEDIA  PIC 9999V99.
77  MEDIAGRUPO PIC 99V99.
77  NOTAEDI    PIC Z9.99.
PROCEDURE DIVISION.
INICIO.
    DISPLAY '       LISTADO DE CALIFICACIONES' UPON PRINTER.
    DISPLAY '       -------------------------' UPON PRINTER.
    DISPLAY ' ' UPON PRINTER.
    DISPLAY 'NOMBRE DEL ALUMNO        NOTA MEDIA' UPON PRINTER.
    DISPLAY '----------------         ----------' UPON PRINTER.
    MOVE 0 TO NUMALUMNOS.
    MOVE 0 TO SUMAMEDIA.
BUCLE.
    DISPLAY 'Escriba nombre del alumno o FIN '.
    ACCEPT NOMBRE.
    IF NOMBRE = 'FIN' GO TO FIN-BUCLE.
    ADD 1 TO NUMALUMNOS.
    MOVE 0 TO SUMA.
    PERFORM LECTURA-NOTAS VARYING I FROM 1 BY 1 UNTIL I > 6.
    DIVIDE SUMA BY 6 GIVING MEDIA.
    ADD MEDIA TO SUMAMEDIA.
    MOVE MEDIA TO NOTAEDI.
    DISPLAY NOMBRE, NOTAEDI UPON PRINTER.
    GO TO BUCLE.
FIN-BUCLE.
    DIVIDE SUMAMEDIA BY NUMALUMNOS GIVING MEDIAGRUPO.
    MOVE MEDIAGRUPO TO NOTAEDI.
    DISPLAY ' ' UPON PRINTER.
    DISPLAY '  NOTA MEDIA DEL GRUPO: ', NOTAEDI UPON PRINTER.
    STOP RUN.
LECTURA-NOTAS.
    DISPLAY 'Escriba nota número ', I.
    ACCEPT NOTA.
    ADD NOTA TO SUMA.
```

Codificación Pascal:

```
PROGRAM GENERACIONDEACTAS (INPUT, OUTPUT, LST);
VAR NOMBRE : STRING[25];
    I, NUMALUMNOS : INTEGER;
    NOTA, SUMA, MEDIA, SUMAMEDIA, MEDIAGRUPO : REAL;
BEGIN (*PP*)
  WRITELN(LST, '       LISTADO DE CALIFICACIONES');
  WRITELN(LST, '       -------------------------');
  WRITELN(LST);
  WRITELN(LST, 'NOMBRE DEL ALUMNO        NOTA MEDIA');
  WRITELN(LST, '----------------         ----------');
  NUMALUMNOS := 0;
  SUMAMEDIA := 0;
  WRITE('Escriba nombre del alumno o FIN ');
```

```
  READLN(NOMBRE);
  WHILE NOMBRE <> 'FIN' DO
    BEGIN (*1*)
      NUMALUMNOS := NUMALUMNOS + 1;
      SUMA := 0;
      FOR I := 1 TO 6 DO
        BEGIN (*2*)
          WRITE('Escriba nota número ',I);
          READLN(NOTA);
          SUMA := SUMA + NOTA
        END; (*2*)
      MEDIA := SUMA / 6;
      SUMAMEDIA := SUMAMEDIA + MEDIA;
      WRITELN(LST, NOMBRE, MEDIA);
      WRITE('Escriba nombre del alumno o FIN ');
      READLN(NOMBRE);
    END; (*1*)
  MEDIAGRUPO := SUMAMEDIA / NUMALUMNOS;
  WRITELN(LST);
  WRITELN(LST, ' NOTA MEDIA DEL GRUPO: ', MEDIAGRUPO)
END. (*PP*)
```

4.4. EJERCICIOS RESUELTOS

E.4.1. Programa que lea una frase en una línea, y cuente su número de vocales.

La frase se lee en una variable alfanumérica, y mediante un índice se recorren y examinan todos sus caracteres.

Pseudocódigo:

```
Programa CONTAR VOCALES
Entorno:
  FRASE es alfanumérica
  NV, I son numéricas enteras
Algoritmo:
  leer FRASE
  NV ← 0
  para I de 1 a longitud(FRASE) hacer
    si es vocal el carácter I de FRASE
      entonces NV ← NV + 1
    finsi
  finpara
  escribir "Número de vocales de la frase: ", NV
Finprograma
```

Estructuras de datos internas (tablas)

5.0. INTRODUCCION

En los capítulos anteriores, los datos manejados en los programas han sido los denominados datos simples (numéricos o alfanuméricos).

En un gran número de problemas es necesario manejar un conjunto de datos, más o menos grande, que están relacionados entre sí, de tal forma que constituyen una unidad para su tratamiento. Por ejemplo, si se quiere manipular una lista de 100 nombres de personas, es conveniente tratar este conjunto de datos de forma unitaria en lugar de utilizar 100 variables, una para cada dato simple.

Un conjunto de datos homogéneos que se tratan como una sola unidad se denomina **estructura de datos**.

Si una estructura de datos reside en la memoria central del computador, se denomina **estructura de datos interna**. Recíprocamente si reside en un soporte externo, se denomina **estructura de datos externa**.

La estructura de datos interna más importante desde el punto de vista de utilización es la **tabla**, que existe en la práctica totalidad de los lenguajes de programación.

Esta estructura se corresponde con los conceptos matemáticos de vector, matriz y poliedro.

5.1. CONCEPTOS Y DEFINICIONES

Una **tabla** consiste en un número fijo, finito y ordenado de elementos, todos del mismo tipo y bajo un **nombre** común para todos ellos.

Se denominan **componentes** a los elementos de una tabla.

La posición de cada componente dentro de la tabla viene determinada por uno o varios **índices**. A cada componente se puede acceder de forma directa indicando sus índices y el nombre de la tabla.

Una tabla se puede estructurar en una, dos o más dimensiones según el número de índices necesarios para acceder a sus elementos. Por lo tanto, la **dimensión** de una tabla es el número de índices que utiliza.

Longitud o **tamaño** de una tabla es el número de componentes que contiene.

El **tipo** de una tabla es el tipo de sus componentes.

Las componentes de una tabla se utilizan de la misma forma que cualquier otra variable de un programa, pudiendo por tanto intervenir en instrucciones de asignación, entrada/salida, etcétera.

Ejemplo:

Tabla que contiene 8 nombres de personas.

ALUMNOS

LUIS	JOSE	ROSA	JUAN	TERE	TOÑI	JAVI	LOLA
1	2	3	4	5	6	7	8

TABLA: La estructura de datos representada.

NOMBRE DE LA TABLA: ALUMNOS.

COMPONENTES: ALUMNOS(1), ALUMNOS(2), ..., ALUMNOS(8).

INDICE: Los números del 1 al 8 que direccionan cada componente.

DIMENSION: Una.

LONGITUD: Ocho.

TIPO: Alfanumérica.

En los diferentes lenguajes de programación hay que declarar las tablas antes de su utilización.

La tabla anterior se declara de la siguiente manera:

BASIC:
```
DIM ALUMNOS$(8)
```

COBOL:
```
01 TABLA-ALUMNOS.
   02 ALUMNOS OCCURS 8 TIMES PIC X(4).
```

Pascal:
```
ALUMNOS : ARRAY[1..8] OF STRING[4];
```

5.2. TIPOS DE TABLAS

Las tablas se clasifican según su dimensión en:

— **Unidimensionales.**
— **Bidimensionales.**
— **Multidimensionales.**

5.2.1. TABLAS UNIDIMENSIONALES

Son tablas de una dimensión. También se denominan **vectores.**

Tienen un solo índice. Cada componente del vector se direcciona mediante su **nombre** seguido del número correspondiente al **índice** entre paréntesis.

El ejemplo anterior nos muestra una tabla de este tipo, en la cual la componente tercera que contiene el valor "ROSA" se denota por:

```
ALUMNOS(3)
```

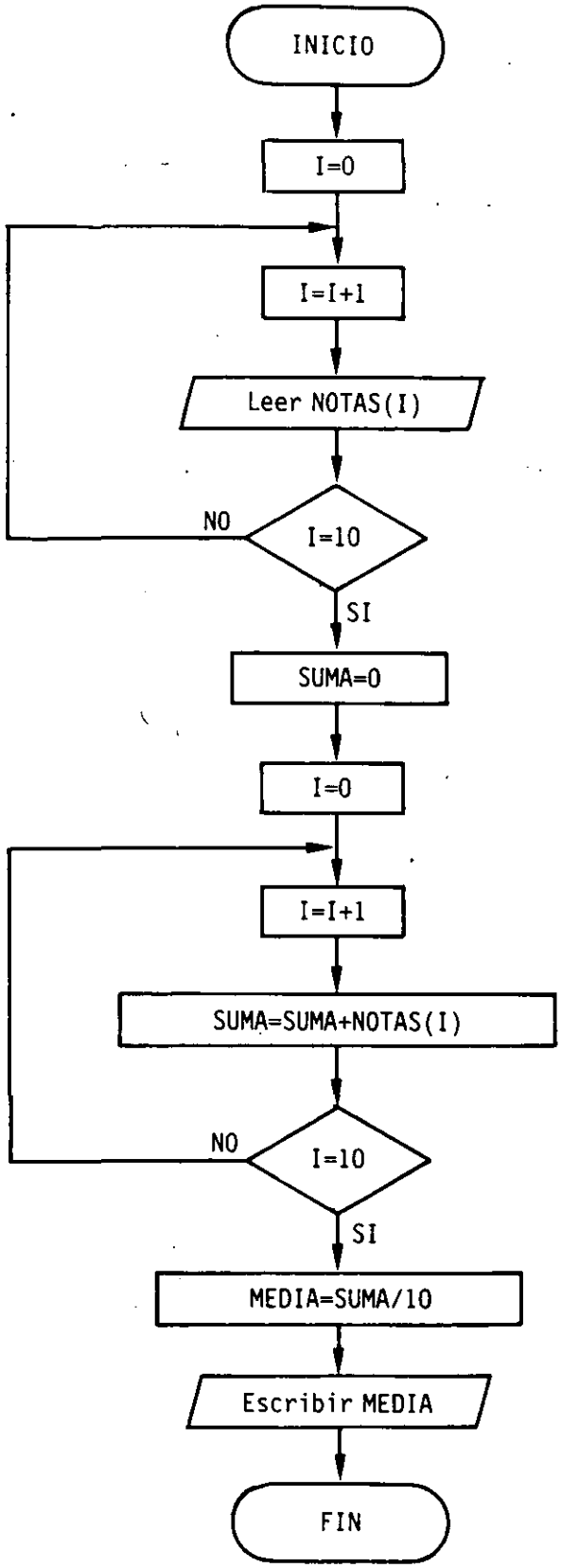
Si queremos intercambiar los contenidos de las componentes primera y segunda, lo haremos utilizando una variable alfanumérica auxiliar AUX de la siguiente manera:

```
AUX ← ALUMNOS(1)
ALUMNOS(1) ← ALUMNOS(2)
ALUMNOS(2) ← AUX
```

Ejercicio:

Programa que lee las calificaciones de un alumno en 10 asignaturas, las almacena en un vector y calcula e imprime su media.

Ordinograma:



Pseudocódigo:

```
Programa NOTA MEDIA
Entorno:
  NOTAS es tabla(10) numérica real
  SUMA, MEDIA son numéricas reales
  I es numérica entera
Algoritmo:
  para I de 1 a 10 hacer
    leer NOTAS(I)
  finpara
  SUMA ← 0
  para I de 1 a 10 hacer
    SUMA ← SUMA + NOTAS(I)
  finpara
  MEDIA ← SUMA / 10
  escribir "Nota media: ", MEDIA
Finprograma
```

Codificación BASIC:

```
10 REM NOTA MEDIA
20 CLS
30 DIM NOTAS(10)
40 FOR I = 1 TO 10
50   PRINT "Nota n° "; I;
60   INPUT NOTAS(I)
70 NEXT I
80 SUMA = 0
90 FOR I = 1 TO 10
100 SUMA = SUMA + NOTAS(I)
110 NEXT I
120 MEDIA = SUMA / 10
130 PRINT "Nota media: "; MEDIA
140 END
```

Codificación COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. NOTA-MEDIA.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  TABLA.
    02 NOTAS OCCURS 10 TIMES PIC 99V99.
77  SUMA  PIC 99V99.
77  MEDIA PIC 99V99.
77  SALIDA PIC Z9.99.
77  I PIC 99.
PROCEDURE DIVISION.
INICIO.
    PERFORM LECTURA VARYING I FROM 1 BY 1 UNTIL I > 10.
    MOVE 0 TO SUMA.
    PERFORM CALCULO VARYING I FROM 1 BY 1 UNTIL I > 10.
```

```
    DIVIDE SUMA BY 10 GIVING MEDIA.
    MOVE MEDIA TO SALIDA.
    DISPLAY 'Nota media: ', SALIDA.
FIN.
    STOP RUN.
LECTURA.
    DISPLAY 'Nota n° ', I.
    ACCEPT NOTAS(I).
CALCULO.
    ADD NOTAS(I) TO SUMA.
```

Codificación Pascal:

```
PROGRAM NOTAMEDIA (INPUT,OUTPUT);
VAR NOTAS : ARRAY[1..10] OF REAL;
    SUMA, MEDIA : REAL;
    I : INTEGER;
BEGIN (*PP*)
  FOR I := 1 TO 10 DO BEGIN (*1*)
    WRITE('Nota n° ', I);
    READLN(NOTAS[I])  END; (*1*)
  SUMA := 0;
  FOR I := 1 TO 10 DO
    SUMA := SUMA + NOTAS[I];
  MEDIA := SUMA / 10;
  WRITE('Nota media: ', MEDIA)
END.(*PP*)
```

5.2.2. TABLAS BIDIMENSIONALES

Son tablas de dos dimensiones. También se denominan **matrices**.

Tienen dos índices, por lo cual cada componente de la matriz se direcciona mediante su **nombre** seguido de los dos **índices** separados por coma y entre paréntesis.

Ejemplo:

Matriz de 6 filas y 8 columnas conteniendo el número de alumnos matriculados en cada grupo de un centro docente por asignatura. Las filas corresponden a los grupos y las columnas a las asignaturas.

MATRICULA

1	35	30	32	32	34	35	34	28
2	40	33	40	37	36	39	40	29
3	25	23	26	21	24	24	25	15
4	33	33	33	32	34	30	32	20
5	45	44	45	44	43	40	44	33
6	24	20	22	22	24	25	24	12
	1	2	3	4	5	6	7	8

NOMBRE DE LA TABLA: MATRICULA.

COMPONENTES: MATRICULA(1,1), MATRICULA(1,2), ..., MATRICULA(6,7), MATRICULA(6,8).

INDICES: Los números del 1 al 6 para las filas y los números 1 al 8 para las columnas.

DIMENSION: Dos.

LONGITUD: 6 * 8 = 48.

TIPO: Numérica entera.

La componente MATRICULA(4,6) almacena el número de alumnos matriculados en el grupo número 4 en la asignatura número 6, que en el ejemplo representado son 30 alumnos.

La tabla anterior se declara:

```
BASIC: DIM MATRICULA(6,8)

COBOL: 01 TABLA-MATRICULA.
          02 GRUPO OCCURS 6 TIMES.
             03 MATRICULA PIC 99 OCCURS 8 TIMES.

Pascal: MATRICULA : ARRAY[1..6,1..8] OF INTEGER;
```

Ejercicio:

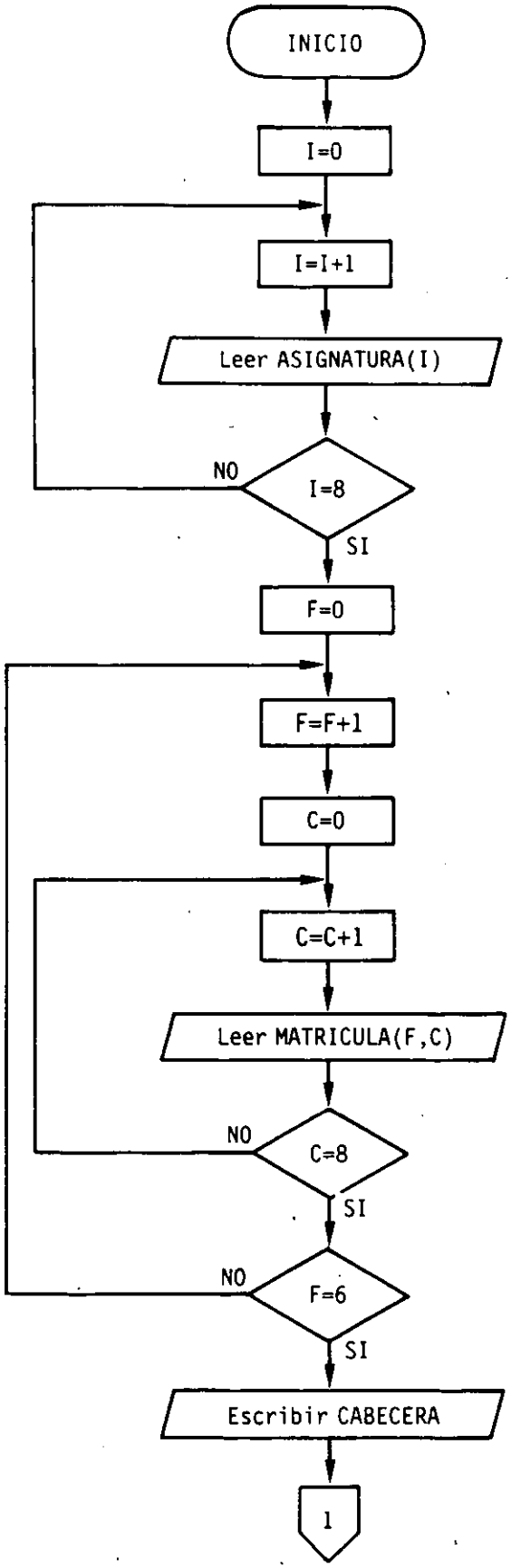
Programa que carga la tabla del ejemplo anterior, y a continuación calcula e imprime el total de alumnos matriculados por asignatura.

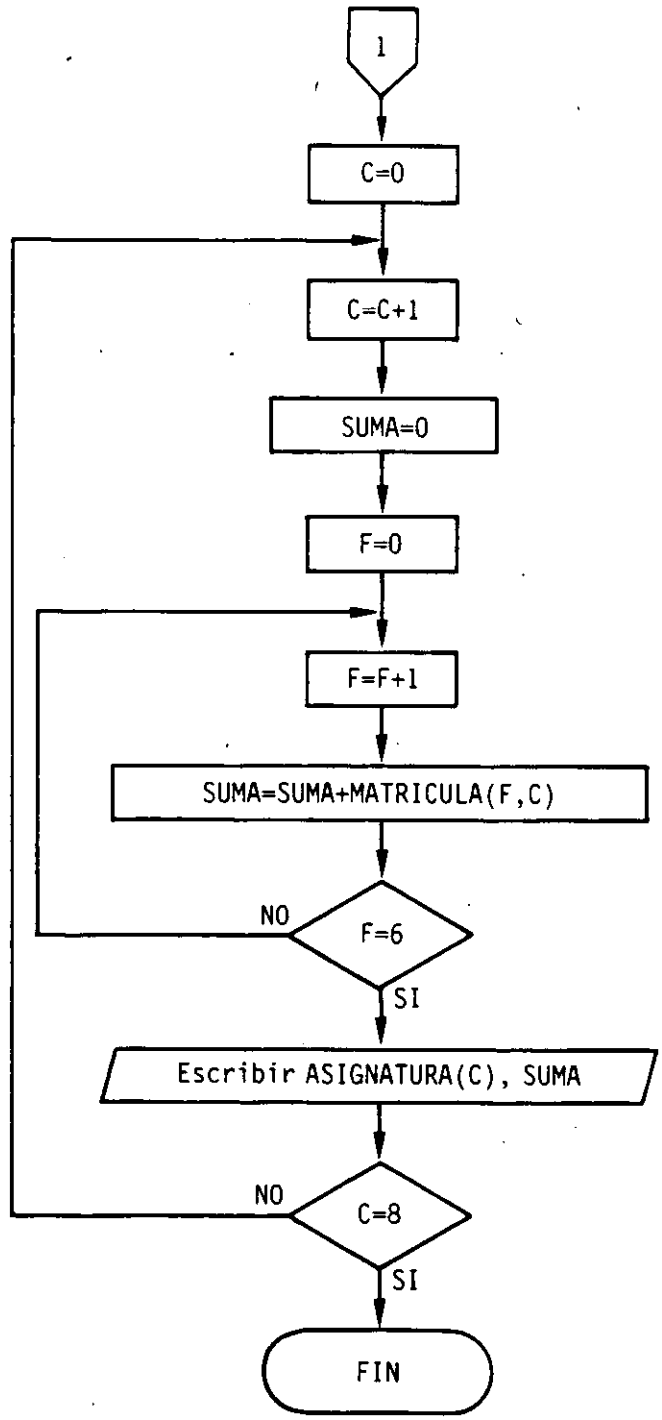
El nombre de las asignaturas es el siguiente:

Asignatura n.° 1. Matemáticas.
Asignatura n.° 2. Lengua Española.
Asignatura n.° 3. Formación Humanística.
Asignatura n.° 4. Ciencias Naturales.
Asignatura n.° 5. Inglés.
Asignatura n.° 6. Informática Básica.
Asignatura n.° 7. Estructura de la Información.
Asignatura n.° 8. Metodología de la Programación.

Suponemos que los 6 grupos son del mismo nivel y tienen las mismas asignaturas.

Ordinograma:





Pseudocódigo:

```
Programa MATRICULACION
Entorno:
  MATRICULA es tabla(6,8) numérica entera
  SUMA, I, F, C son numéricas enteras
  ASIGNATURA es tabla(8) alfanumérica
```

```
Algoritmo:
  para I de 1 a 8 hacer
    leer ASIGNATURA(I)
  finpara
  para F de 1 a 6 hacer
    para C de 1 a 8 hacer
      leer MATRICULA(F,C)
    finpara
  finpara
  escribir "Alumnos matriculados"
  escribir "ASIGNATURA", "NUM. ALUMNOS"
  para C de 1 a 8 hacer
    SUMA ← 0
    para F de 1 a 6 hacer
      SUMA ← SUMA + MATRICULA(F,C)
    finpara
    escribir ASIGNATURA(C), SUMA
  finpara
Finprograma
```

Codificación BASIC:

```
10 REM MATRICULACION
20 CLS : DIM MATRICULA(6,8), ASIGNATURA$(8)
30 FOR I = 1 TO 8
40   READ ASIGNATURA$(I)
50 NEXT I
60 FOR F = 1 TO 6
70   FOR C = 1 TO 8
80     PRINT "GRUPO: "; F; " ASIGNATURA: "; ASIGNATURA$(C);
90     INPUT MATRICULA(F,C)
100  NEXT C
110 NEXT F
120 PRINT "Alumnos matriculados"
130 PRINT "ASIGNATURA", "NUM. ALUMNOS"
140 PRINT "==========", "==========="
150 FOR C = 1 TO 8
160   SUMA = 0
170   FOR F = 1 TO 6
180     SUMA = SUMA + MATRICULA(F,C)
190   NEXT F
200   PRINT ASIGNATURA$(C), SUMA
210 NEXT C
220 DATA Matemáticas, Lengua Española, Formación Humanística,
        Ciencias Naturales, Inglés, Informática Básica,
        Estructura de la Información, Metodología de la Programación
230 END
```

Codificación COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MATRICULACION.
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
01  TABLA-MATRICULA.
    02 GRUPO OCCURS 6 TIMES.
       03 MATRICULA PIC 99 OCCURS 8 TIMES.
01  TABLA-ASIGNATURAS.
    02 FILLER PIC X(11) VALUE 'Matemáticas'.
    02 FILLER PIC X(11) VALUE 'Lengua Esp.'.
    02 FILLER PIC X(11) VALUE 'Form.Human.'.
    02 FILLER PIC X(11) VALUE 'Ciencias N.'.
    02 FILLER PIC X(11) VALUE 'Inglés     '.
    02 FILLER PIC X(11) VALUE 'Inf. Básica'.
    02 FILLER PIC X(11) VALUE 'Estr.Infor.'.
    02 FILLER PIC X(11) VALUE 'Metod.Prog.'.
01  TABLA-ASIG REDEFINES TABLA-ASIGNATURAS.
    02 ASIGNATURA PIC X(11) OCCURS 8 TIMES.
77  SUMA PIC 999.
77  I PIC 9.
77  F PIC 9.
77  C PIC 9.
PROCEDURE DIVISION.
INICIO.
    PERFORM FILA VARYING F FROM 1 BY 1 UNTIL F > 6.
    DISPLAY 'Alumnos matriculados'.
    DISPLAY 'ASIGNATURA ', 'NUM. ALUMNOS'.
    DISPLAY '========== ', '============'.
    PERFORM COL VARYING C FROM 1 BY 1 UNTIL C > 8.
FIN.
    STOP RUN.
FILA.
    PERFORM COLU VARYING C FROM 1 BY 1 UNTIL C > 8.
COLU.
    DISPLAY 'Grupo: ', F, ' Asignatura: ', ASIGNATURA(C).
    ACCEPT MATRICULA(F, C).
COL.
    MOVE 0 TO SUMA.
    PERFORM FIL VARYING F FROM 1 BY 1 UNTIL F > 6.
    DISPLAY ASIGNATURA(C), SUMA.
FIL.
    ADD MATRICULA(F, C) TO SUMA.
```

Codificación Pascal:

```
PROGRAM MATRICULACION (INPUT,OUTPUT);
TYPE STRING = PACKED ARRAY[1..11] OF CHAR;
VAR MATRICULA : ARRAY[1..6,1..8] OF INTEGER;
    ASIGNATURA : ARRAY[1..8] OF STRING;
    SUMA, I, F, C : INTEGER;
BEGIN (*PP*)
  ASIGNATURA[1] := 'Matemáticas';
  ASIGNATURA[2] := 'Lengua Esp.';
  ASIGNATURA[3] := 'Form.Human.';
  ASIGNATURA[4] := 'Ciencias N.';
  ASIGNATURA[5] := 'Inglés     ';
  ASIGNATURA[6] := 'Inf. Básica';
```

```
    ASIGNATURA[7] := 'Estr.Infor.';
    ASIGNATURA[8] := 'Metod.Prog.';
    FOR F := 1 TO 6 DO
      FOR C := 1 TO 8 DO
        READ(MATRICULA[F,C]);
    WRITE('Alumnos matriculados');
    WRITE('ASIGNATURA    NUM. ALUMNOS');
    WRITE('==========    ============');
    FOR C := 1 TO 8 DO
      BEGIN (*1*)
        SUMA := 0;
        FOR F := 1 TO 6 DO
          SUMA := SUMA + MATRICULA[F,C];
        WRITE(ASIGNATURA[C], '       ', SUMA)
      END (*1*)
  END. (*PP*)
```

5.2.3. TABLAS MULTIDIMENSIONALES

Son tablas de tres o más dimensiones. También se les denomina **poliedros**.

Este tipo de tablas no son de uso frecuente, no obstante es una herramienta útil para un determinado número de problemas.

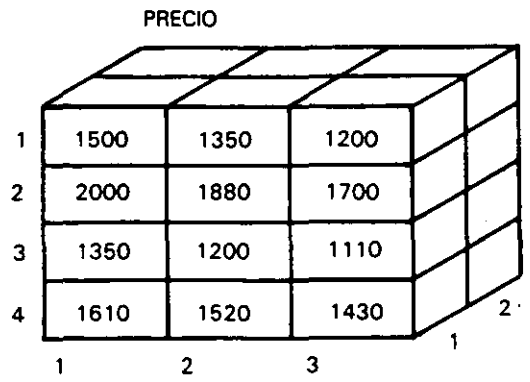
La mayoría de los lenguajes de programación admiten estas estructuras, aunque cada uno de ellos tiene una limitación con respecto al número máximo de dimensiones permitidas.

Ejemplo:

Poliedro de tres dimensiones que consta de 24 componentes agrupados de la siguiente forma:

— Primera dimensión que indica el número de un ARTICULO de 1 a 4.
— Segunda dimensión que indica el grado de CALIDAD de 1 a 3.
— Tercera dimensión que indica el tipo de VENTA de 1 a 2 (1 para venta al por mayor y 2 para venta al detall).

Cada componente almacena el precio para los artículos según la calidad y el tipo de venta.



NOMBRE DE LA TABLA: PRECIO.

COMPONENTES: PRECIO(1,1,1), PRECIO(1,1,2), ..., PRECIO(2,1,1), PRECIO(2,1,2), ..., PRECIO(4,3,1), PRECIO(4,3,2).

INDICES: Los números 1 al 4 para la primera dimensión, del 1 al 3 para la segunda y del 1 al 2 para la tercera.

DIMENSION: Tres.

LONGITUD: 4 * 3 * 2 = 24.

TIPO: Numérica entera

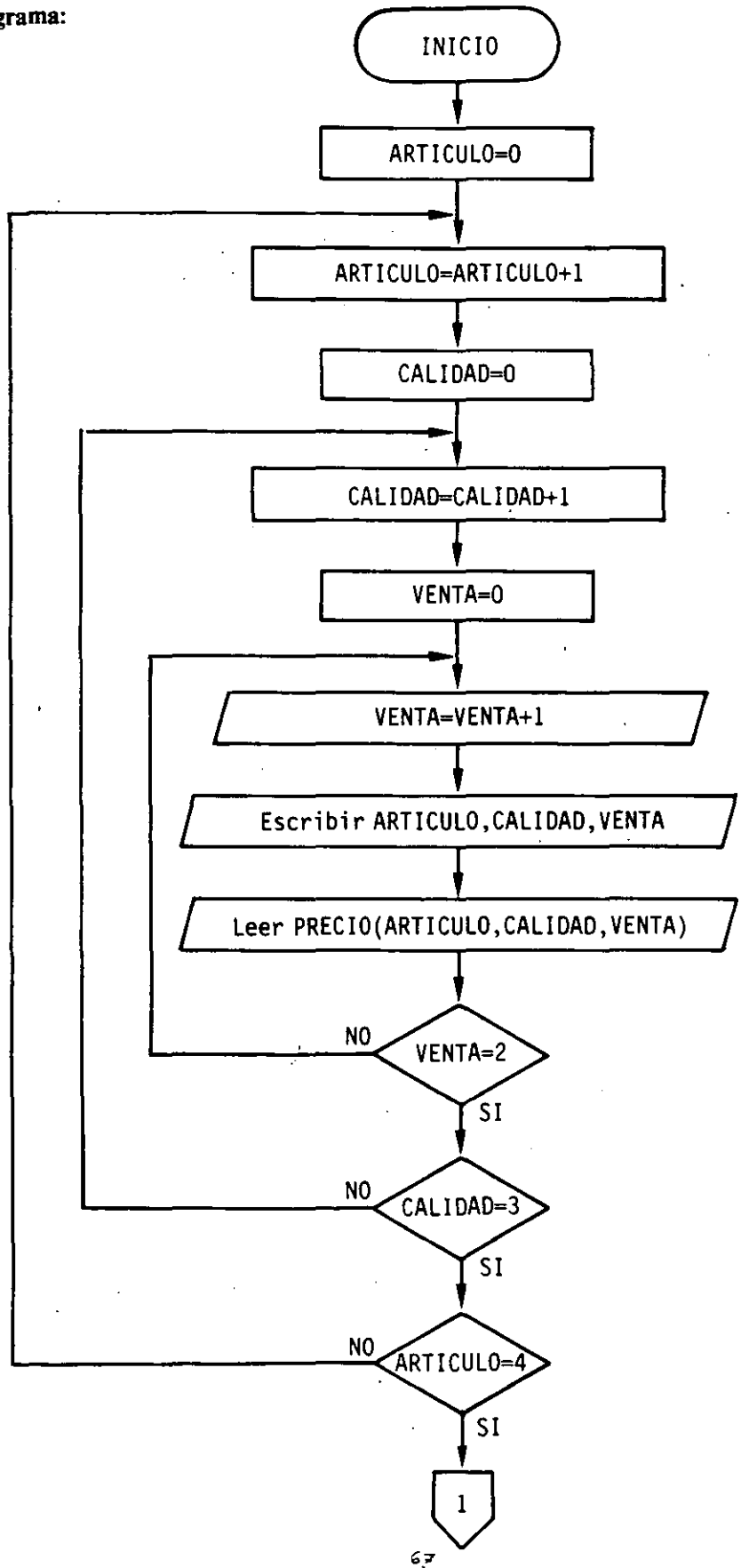
La componente PRECIO(4,3,2) almacena el precio de un ARTICULO del número 4, de CALIDAD 3 que se vende al *Detall*. En el ejemplo representado su valor está oculto.
Las declaraciones son en este caso:

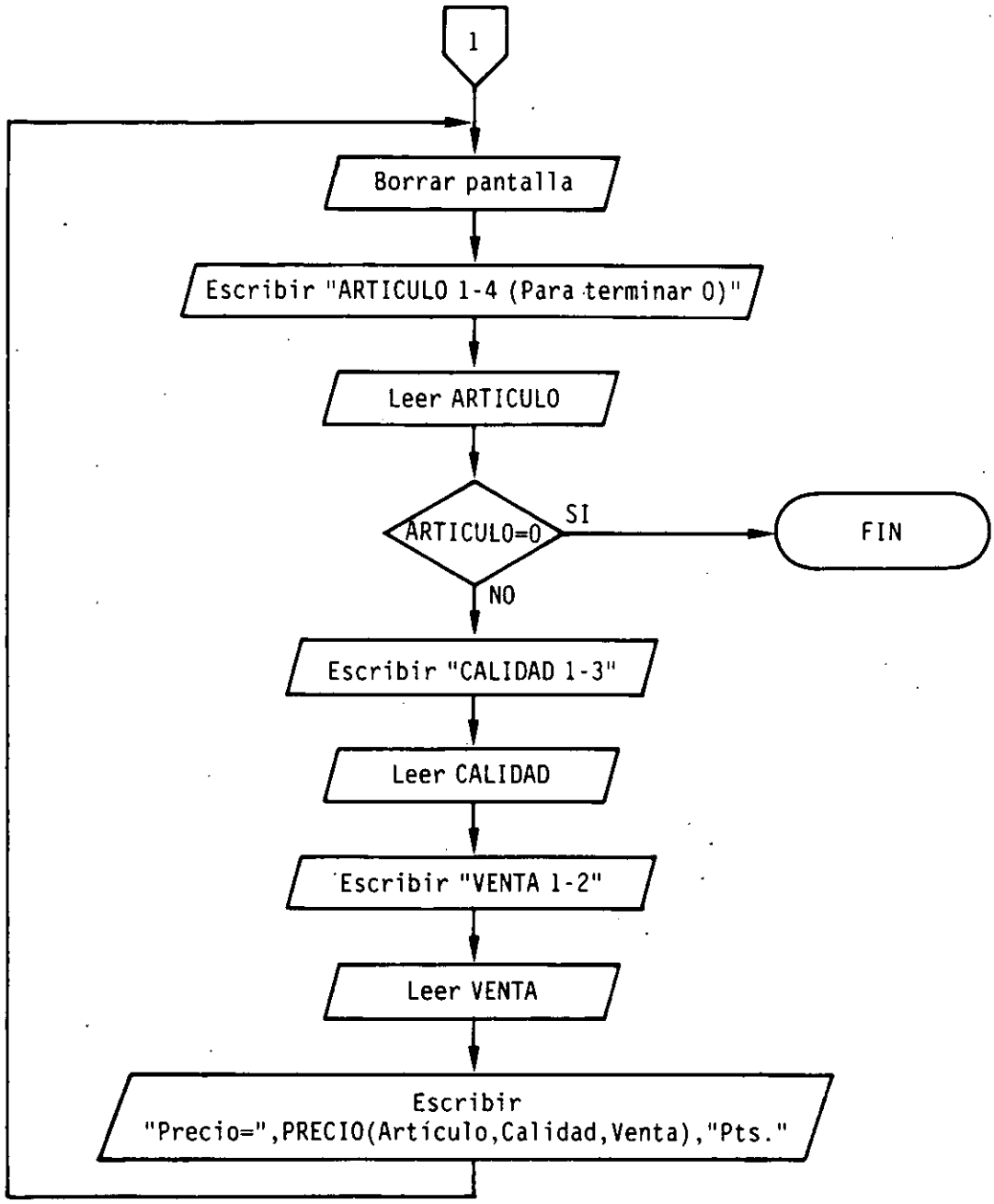
```
BASIC:  DIM PRECIO(4,3,2)
COBOL:  01 TABLA-PRECIO.
           02 ARTICULO OCCURS 4 TIMES.
              03 CALIDAD OCCURS 3 TIMES.
                 04 PRECIO PIC 9999 OCCURS 2 TIMES.
Pascal: PRECIO : ARRAY[1..4,1..3,1..2] OF INTEGER;
```

Ejercicio:

Programa que carga la tabla del ejemplo anterior y posibilita sucesivas consultas de precios introduciendo como datos de entrada el número de artículo, su calidad y el tipo de venta. Para terminar la serie de consultas se introducirá un 0 en el número de artículo.
El programa consta de dos bloques, el primero consiste en la carga de precios en la tabla y el segundo en el acceso sucesivo a las distintas componentes de la tabla solicitadas.

Ordinograma:





Pseudocódigo:

```
Programa CONSULTAS
Entorno:
  PRECIO es tabla(4,3,2) numérica entera
  ARTICULO, CALIDAD, VENTA son numéricas enteras
Algoritmo:
  para ARTICULO de 1 a 4 hacer
    para CALIDAD de 1 a 3 hacer
      para VENTA de 1 a 2 hacer
```

```
        escribir ARTICULO,CALIDAD,VENTA
        leer PRECIO(ARTICULO,CALIDAD,VENTA)
      finpara
    finpara
  finpara
  iterar
    borrar pantalla
    escribir "ARTICULO 1-4 (para terminar 0)"
    leer ARTICULO
    salir si ARTICULO = 0
    escribir "CALIDAD 1-3"
    leer CALIDAD
    escribir "VENTA 1-2"
    leer VENTA
    escribir "PRECIO = ", PRECIO(ARTICULO,CALIDAD,VENTA), " Pts."
  finiterar
Finprograma
```

Codificación BASIC:

```
10 REM CONSULTAS
20 CLS
30 DIM PRECIO(4,3,2)
40 FOR ARTICULO = 1 TO 4
50.  FOR CALIDAD = 1 TO 3
60     FOR VENTA = 1 TO 2
70       PRINT ARTICULO, CALIDAD, VENTA
80       INPUT PRECIO(ARTICULO,CALIDAD,VENTA)
90     NEXT VENTA
100  NEXT CALIDAD
110 NEXT ARTICULO
120 CLS
130 PRINT "ARTICULO 1-4 (para terminar 0)";
140 INPUT ARTICULO
150 IF ARTICULO = 0 THEN GOTO 200
160 INPUT "CALIDAD 1-3 "; CALIDAD
170 INPUT "VENTA 1-2 "; VENTA
180 PRINT "PRECIO = "; PRECIO(ARTICULO,CALIDAD,VENTA); " Pts."
190 GOTO 130
200 END
```

Codificación COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CONSULTAS.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  TABLA-PRECIO.
    02 ARTICULO OCCURS 4 TIMES.
       03 CALIDAD OCCURS 3 TIMES.
          04 PRECIO PIC 9999 OCCURS 2 TIMES.
77  ARTICULO PIC 9.
77  CALIDAD  PIC 9.
```

```
77  VENTA    PIC 9.
PROCEDURE DIVISION.
INICIO.
    PERFORM ART VARYING ARTICULO FROM 1 BY 1 UNTIL ARTICULO > 4.
BUCLE-CONSULTAS.
    DISPLAY 'ARTICULO 1-4 (para terminar 0)'.
    ACCEPT ARTICULO.
    IF ARTICULO = 0 THEN GO TO FIN.
    DISPLAY 'CALIDAD 1-3 '.
    ACCEPT CALIDAD.
    DISPLAY 'VENTA 1-2 '.
    ACCEPT VENTA.
    DISPLAY 'PRECIO = ', PRECIO(ARTICULO, CALIDAD, VENTA),
      ' Pts.'.
   GO TO BUCLE-CONSULTAS.
FIN.
    STOP RUN.
ART.
    PERFORM CAL VARYING CALIDAD FROM 1 BY 1 UNTIL CALIDAD > 3.
CAL.
    PERFORM VENT VARYING VENTA FROM 1 BY 1 UNTIL VENTA > 2.
VENT.
    DISPLAY 'Artículo ', 'Calidad ', 'Venta'.
    DISPLAY ARTICULO, CALIDAD, VENTA.
    ACCEPT PRECIO(ARTICULO, CALIDAD, VENTA).
```

Codificación Pascal:

```
PROGRAM CONSULTAS (INPUT,OUTPUT);
VAR PRECIO : ARRAY[1..4,1..3,1..2] OF INTEGER;
    ARTICULO,CALIDAD,VENTA : INTEGER;
BEGIN (*PP*)
  FOR ARTICULO := 1 TO 4 DO
    FOR CALIDAD := 1 TO 3 DO
      FOR VENTA := 1 TO 2 DO
        BEGIN (*1*)
          WRITELN(ARTICULO,CALIDAD,VENTA);
          READLN(PRECIO[ARTICULO,CALIDAD,VENTA])
        END; (*1*)
  WRITE('ARTICULO 1-4 (para terminar 0)');
  READLN(ARTICULO);
  WHILE ARTICULO <> 0 DO
    BEGIN (*2*)
       WRITE('CALIDAD 1-3 ');
       READLN(CALIDAD);
       WRITE('VENTA 1-2 ');
       READLN(VENTA);
       WRITELN('PRECIO = ', PRECIO[ARTICULO,CALIDAD,VENTA],
         ' Pts.');
       WRITE('ARTICULO 1-4 (para terminar 0)');
       READLN(ARTICULO)
     END (*2*)
 END. (*PP*)
```

Búsqueda y clasificación interna

6.0. INTRODUCCION

Existen multitud de algoritmos para manejar las estructuras de datos internas, de entre ellos destacan por su importancia y frecuencia de utilización los de búsqueda y clasificación.

Estos algoritmos son clásicos y su conocimiento es fundamental para cualquier programador.

A lo largo de la historia de la programación se han estudiado y propuesto diversos métodos. De todos ellos propondremos los más importantes y también los más sencillos.

La elección final de uno u otro método dependerá de las características particulares de un problema concreto y de sus datos.

6.1. BUSQUEDA LINEAL

Dada una tabla y un valor del mismo tipo que sus componentes, la búsqueda consiste en determinar si ese valor está en la tabla y qué posición ocupa.

6.1.1. BUSQUEDA LINEAL EN UN VECTOR

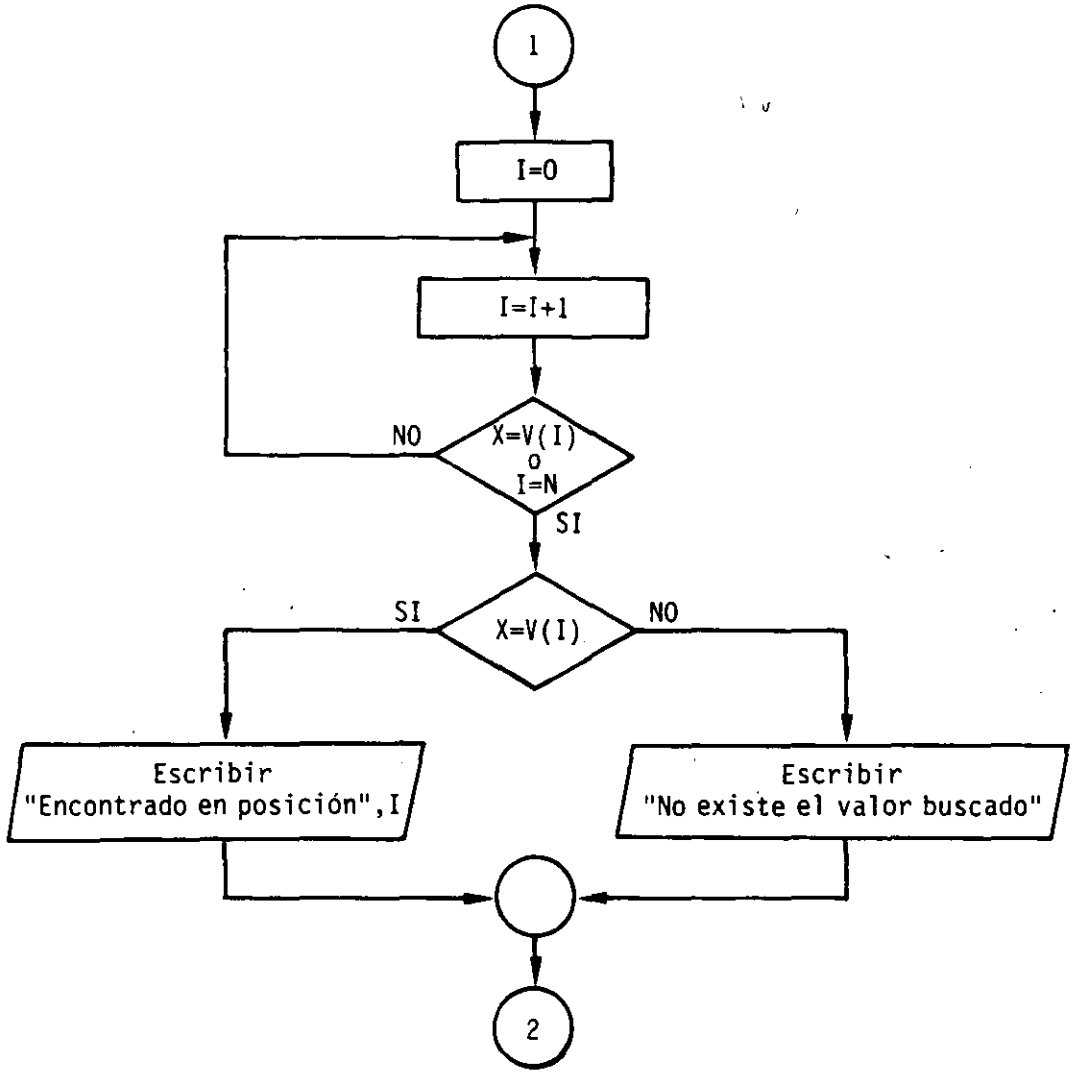
Se recorre el vector de izquierda a derecha hasta encontrar una componente cuyo valor coincida con el buscado o hasta que se acabe el vector. En este último caso, el algoritmo debe indicar la no existencia de dicho valor.

En los casos en que pueda aparecer el valor repetido, el algoritmo indicará el de índice menor. Con una pequeña modificación podemos obtener las distintas posiciones que ocupa el valor.

Este método es válido tanto para vectores desordenados como ordenados, aunque para vectores ordenados veremos otros métodos más eficientes.

Sea un vector V de N componentes y un valor X a buscar en V:

Ordinograma:



Pseudocódigo:

```
...
I ← 0
iterar
  I ← I + 1
  salir si x = V(I) o I = N
finiterar
si X = V(I)
  entonces escribir "Encontrado en posición ", I
  sino escribir "No existe el valor buscado"
finsi
...
```

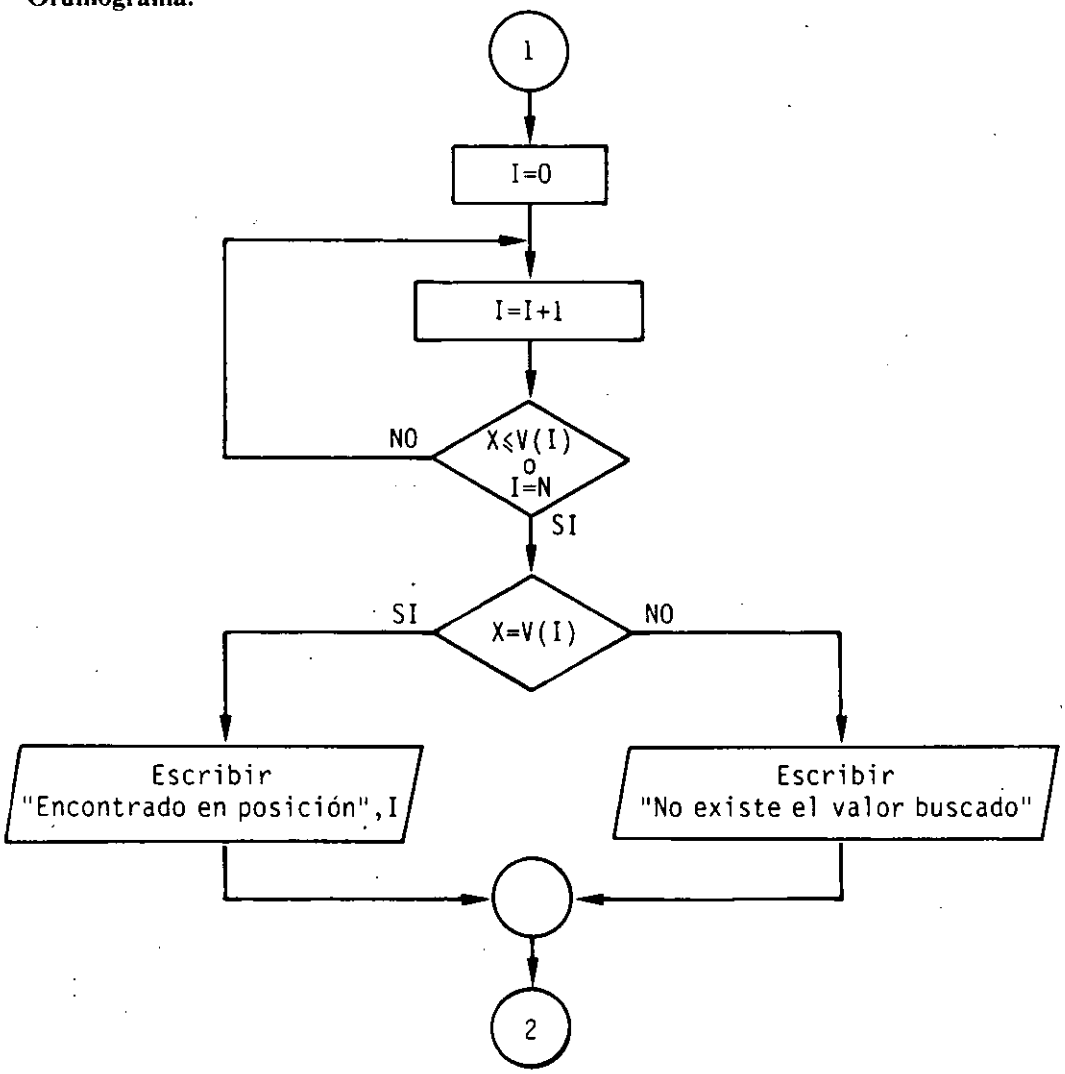
6.1.2. BUSQUEDA LINEAL EN UN VECTOR ORDENADO

Cuando el vector de búsqueda está ordenado se consigue un algoritmo más eficiente sin más que modificar la codificación de terminación en el algoritmo anterior.

La ventaja que se obtiene es en el caso de orden ascendente, una vez sobrepasado el valor buscado, no es necesario recorrer el resto del vector para saber que el valor no existe.

Sea un vector V de N componentes clasificado en orden ascendente y un valor X a buscar:

Ordinograma:



Pseudocódigo:

```
...
I ← 0
iterar
  I ← I + 1
  salir si X <= V(I) o I = N
finiterar
si X = V(I)
  entonces escribir "Encontrado en posición ", I
  sino escribir "No existe el valor buscado"
finsi
...
```

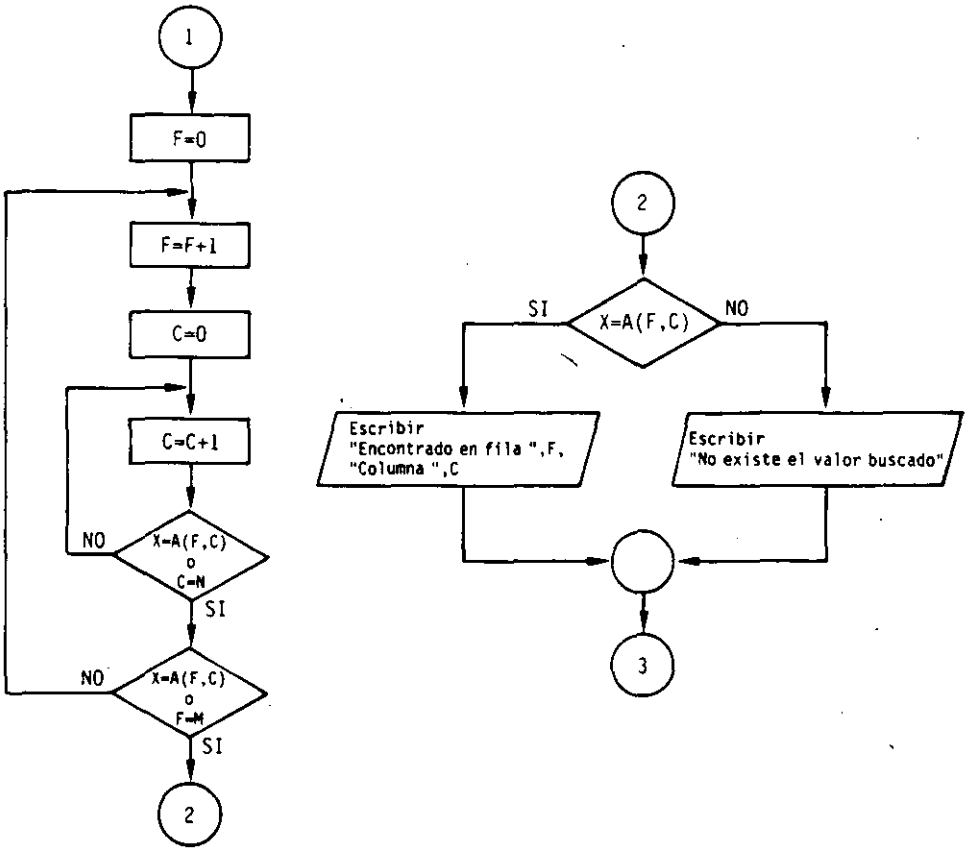
6.1.3. BUSQUEDA LINEAL EN UNA MATRIZ

Se realiza mediante el anidamiento de dos bucles de búsqueda HASTA(UNTIL) cuya finalización vendrá dada por la aparición del valor buscado o la terminación de la matriz.

Usualmente se comienza recorriendo la matriz por filas, aunque cambiando de posición los índices con sus correspondientes límites se puede hacer igualmente por columnas.

Sea una matriz A de M filas y N columnas y un valor X a buscar en A:

Ordinograma:



Pseudocódigo:

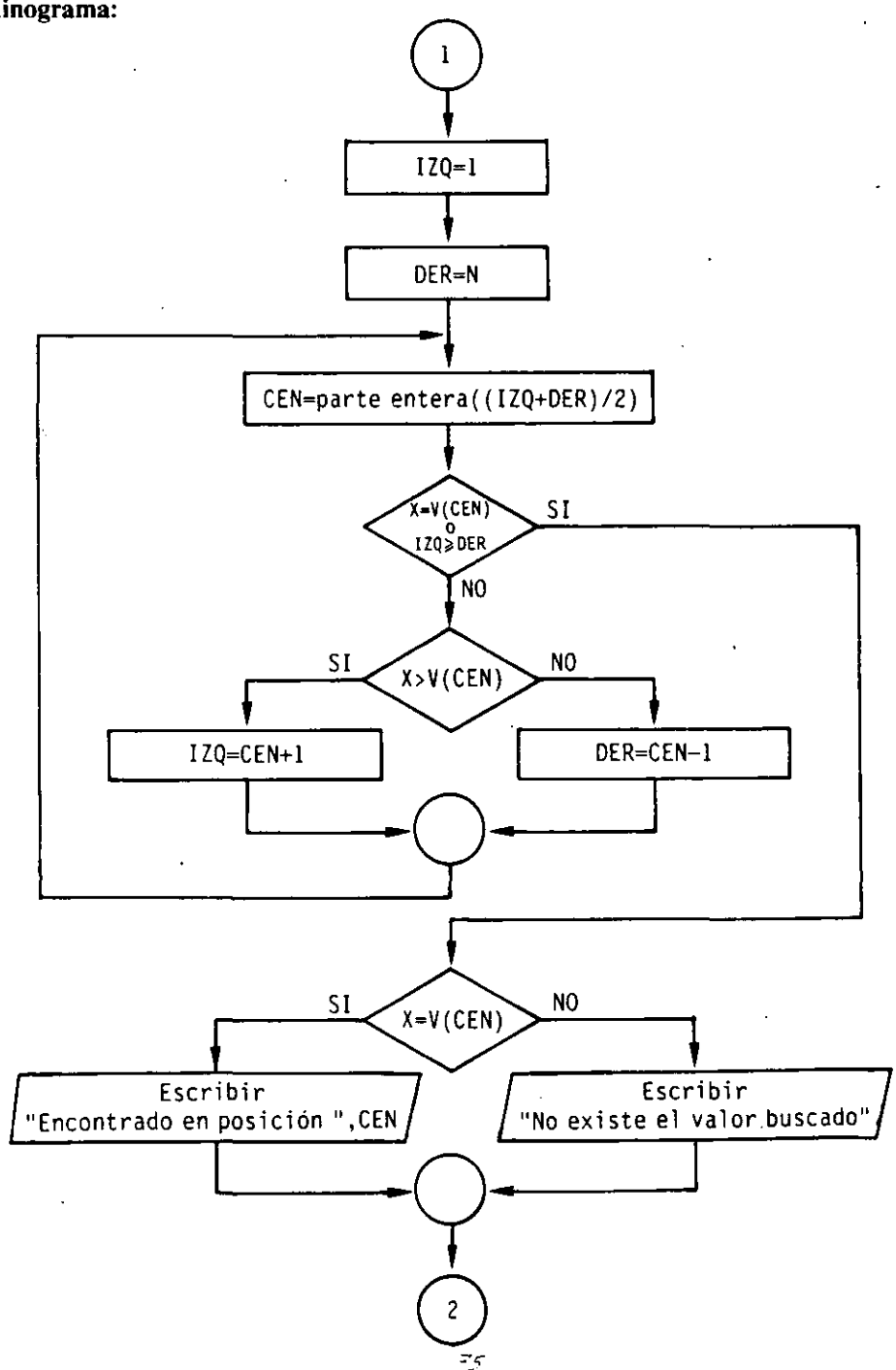
```
...
F ← 0
iterar
  F ← F + 1
  C ← 0
  iterar
    C ← C + 1
    salir si X = A(F,C) o C = N
  finiterar
  salir si X = A(F,C) o F = M
finiterar
si X = A(F,C)
  entonces escribir "Encontrado en fila ", F, " Columna ", C
  sino escribir "No existe el valor buscado"
finsi
...
```

6.2. BUSQUEDA BINARIA O DICOTOMICA

Este algoritmo es válido exclusivamente para vectores ordenados y consiste en comparar en primer lugar con la componente central del vector, y si no es igual al valor buscado se reduce el intervalo de búsqueda a la mitad derecha o izquierda según donde pueda encontrarse el valor a buscar. El algoritmo termina si se encuentra el valor buscado o si el tamaño del intervalo de búsqueda queda anulado.

En los casos en que existan repeticiones en el vector, del valor buscado, este algoritmo obtendrá uno de ellos aleatoriamente según los lugares que ocupen, los cuales necesariamente son consecutivos.

Ordinograma:



Pseudocódigo:

```
...
IZQ ← 1
DER ← N
iterar
  CEN ← parte-entera((IZQ + DER) / 2)
  salir si X = V(CEN) o IZQ >= DER
  si X > V(CEN)
    entonces IZQ ← CEN + 1
    sino DER ← CEN - 1
  finsi
finiterar
si X = V(CEN)
  entonces escribir "Encontrado en posición ", CEN
  sino escribir "No existe el valor buscado"
finsi
...
```

6.3. ORDENACION DE TABLAS

Muchos algoritmos necesitan utilizar tablas ordenadas, por lo cual es preciso previamente ordenarlas o clasificarlas.

Podemos definir una **ordenación** como la reagrupación de un conjunto de elementos en una secuencia específica.

Los tipos de ordenación que realizaremos son:

— Ordenación ascendente o creciente.
  Consiste en situar los valores mayores a la derecha y los menores a la izquierda. Los valores repetidos quedarán en posiciones consecutivas. En casos de valores alfanuméricos el orden utilizado es el lexicográfico.
— Ordenación descendente o decreciente.
  Es la ordenación inversa a la anterior.

Estudiaremos varios métodos de ordenación aplicados a vectores teniendo en cuenta que se pueden generalizar a matrices con respecto a una de sus filas o columnas y a poliedros con respecto a una de sus dimensiones.

6.3.1. ORDENACION POR INSERCION DIRECTA

También se denomina **método de la baraja.**

Supongamos un vector V de N componentes numéricas o alfanuméricas que deseamos clasificar en orden ascendente.

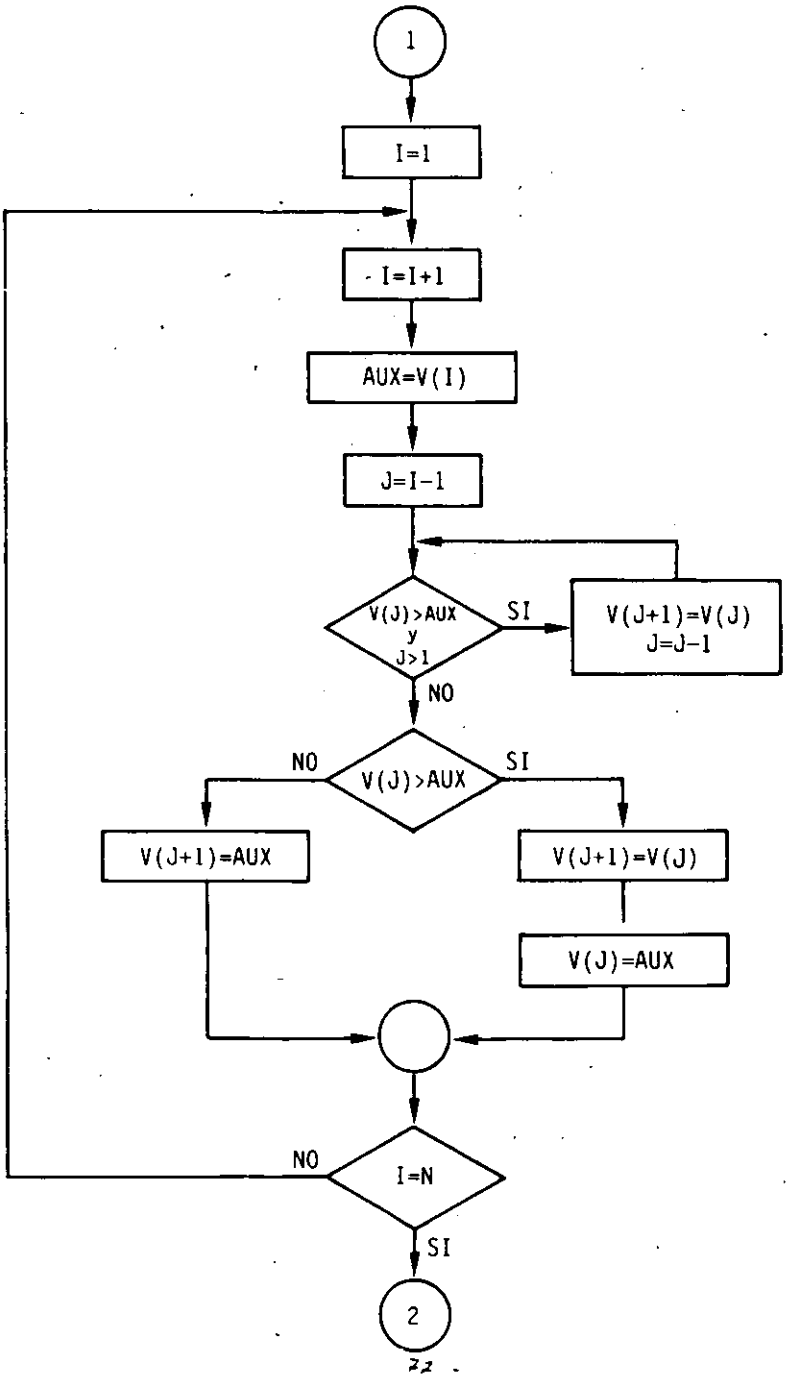
El método consiste en repetir el siguiente proceso desde la segunda componente hasta la última: Se toma dicha componente y se inserta en el lugar que le corresponda entre las componentes situadas a su izquierda (éstas ya estarán ordenadas). Las componentes superiores a la tratada se desplazarán un lugar a la derecha.

Ejemplo:

Veamos la evolución de una secuencia de números mediante este método.

Secuencia inicial	3	2	4	1	2
Primer paso	2	3	4	1	2
Segundo paso	2	3	4	1	2
Tercer paso	1	2	3	4	2
Cuarto paso	1	2	2	3	4

Ordinograma:



Pseudocódigo:

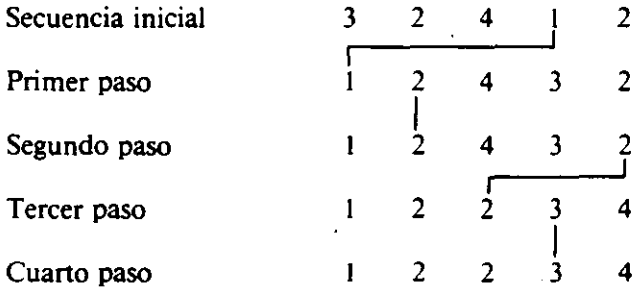
```
...
para I de 2 a N hacer
  AUX ← V(I)
  J ← I - 1
  mientras V(J) > AUX y J > 1 hacer
    V(J + 1) ← V(J)
    J ← J - 1
  finmientras
  si V(J) > AUX
    entonces
      V(J + 1) ← V(J)
      V(J) ← AUX
    sino V(J + 1) ← AUX
  finisi
finpara
...
```

6.3.2. ORDENACION POR SELECCION DIRECTA

El método consiste en repetir el siguiente proceso desde el primer elemento hasta el penúltimo: Se selecciona la componente de menor valor de todas las situadas a la derecha de la tratada y se intercambia con ésta.

Ejemplo:

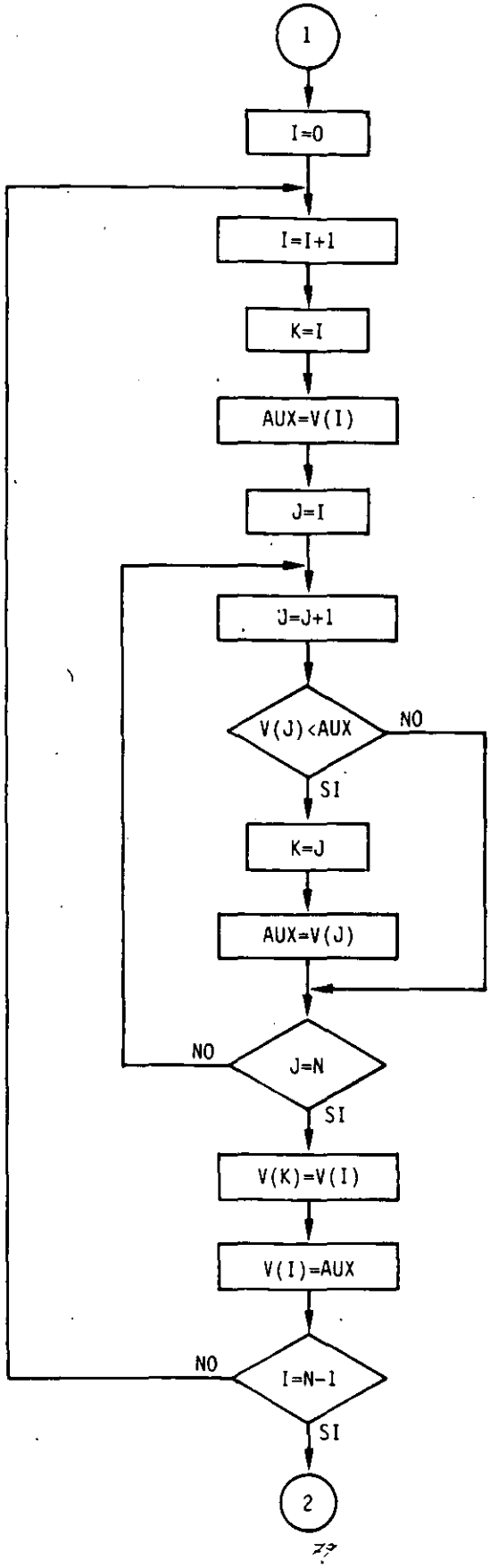
Evolución de una secuencia de números.



Pseudocódigo:

```
...
para I de 1 a N - 1 hacer
  K ← I
  AUX ← V(I)
  para J de I + 1 a N hacer
    si V(J) < AUX
      entonces
        K ← J
        AUX ← V(J)
    finsi
  finpara
  V(K) ← V(I)
  V(I) ← AUX
finpara
...
```

Ordinograma:



6.3.3. ORDENACION POR INTERCAMBIO DIRECTO. METODO DE LA BURBUJA

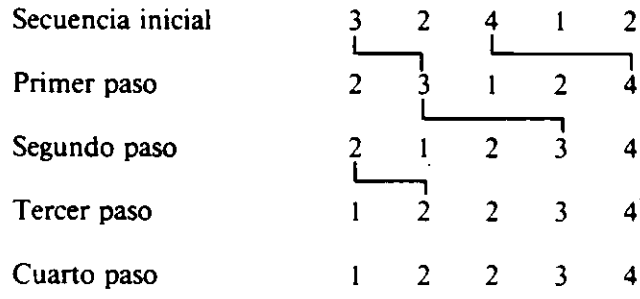
Este método tiene dos versiones basadas en la misma idea que consiste en recorrer sucesivamente el vector comparando los elementos consecutivos e intercambiándolos cuando estén descolocados. El recorrido del vector se puede hacer de izquierda a derecha (desplazando los valores mayores hacia su derecha) o de derecha a izquierda (desplazando los valores menores hacia su izquierda), ambos para la clasificación en orden ascendente.

6.3.3.1. Recorrido izquierda-derecha

Esta versión del método va colocando en cada pasada el mayor elemento de los tratados en la última posición quedando colocado y por lo tanto excluido de los elementos a tratar en la siguiente pasada.

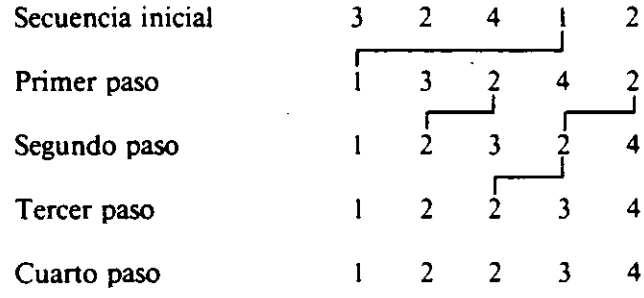
Ejemplo:

Evolución de una secuencia de números con esta versión.

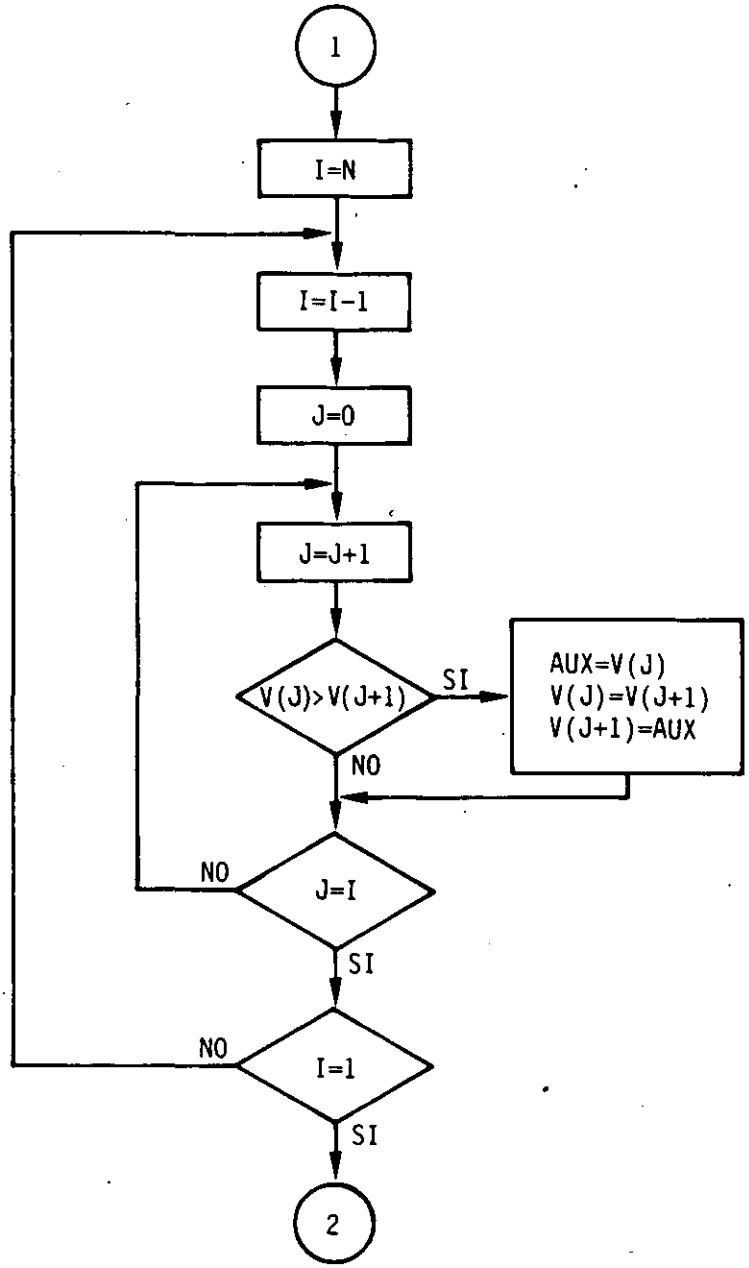


6.3.3.2. Recorrido derecha-izquierda

Esta versión es simétrica a la anterior desplazando los elementos menores hacia la izquierda.



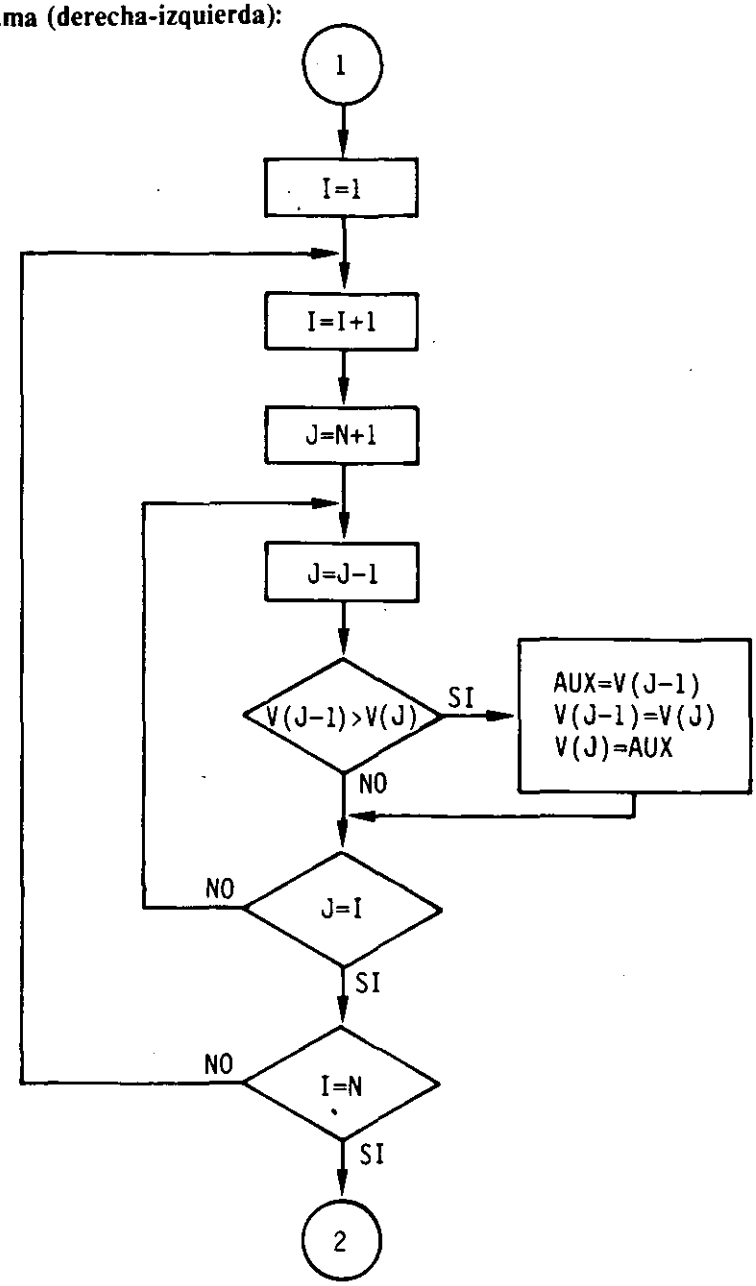
Ordinograma (izquierda-derecha):



Pseudocódigo (izquierda-derecha):

```
...
para I de N - 1 a 1 con incremento -1 hacer
  para J de 1 a I hacer
    si V(J) > V(J + 1)
      entonces
        AUX ← V(J)
        V(J) ← V(J + 1)
        V(J + 1) ← AUX
    finsi
  finpara
finpara
...
```

Ordinograma (derecha-izquierda):



Pseudocódigo (derecha-izquierda):

```
...
para I de 2 a N hacer
  para J de N a I con incremento -1 hacer
    si V(J - 1) > V(J)
      entonces
        AUX ← V(J - 1)
        V(J - 1) ← V(J)
        V(J) ← AUX
    finsi
  finpara
finpara
...
```

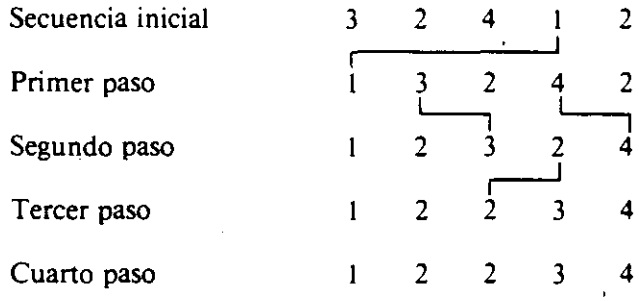
6.3.4. ORDENACION POR INTERCAMBIO DIRECTO. METODO DE LA SACUDIDA

Consiste en mezclar las dos versiones del método de la burbuja alternativamente, de tal forma que se realiza una pasada de derecha a izquierda y a continuación otra de izquierda a derecha, recortándose los elementos a tratar por ambos lados del vector.

Este método es más eficiente que los anteriores en la mayoría de los casos.

Ejemplo:

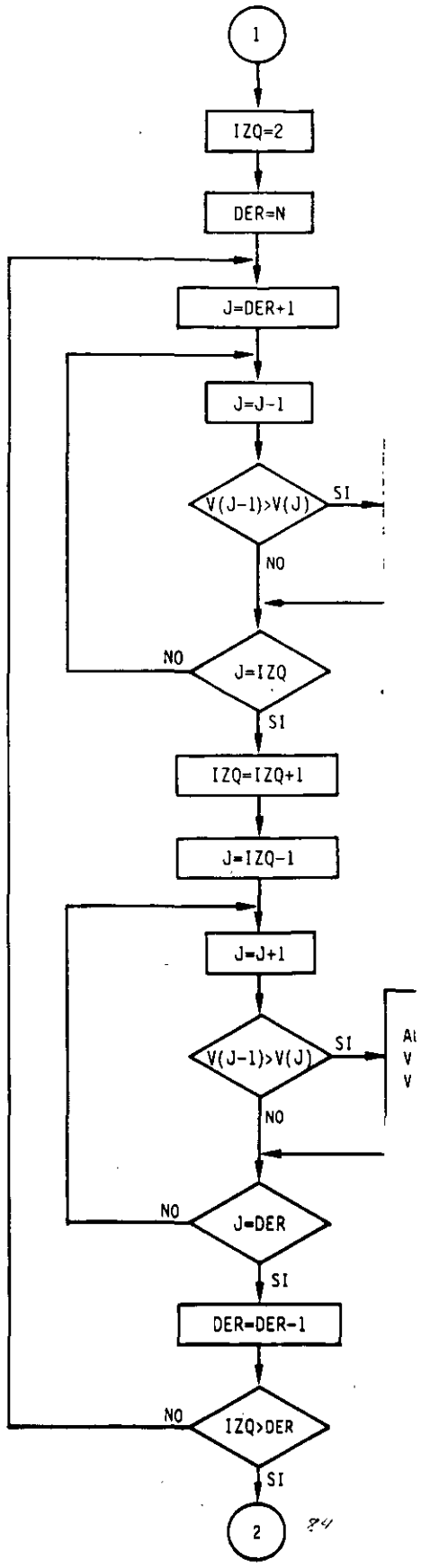
Evolución de una secuencia de números mediante el método de la sacudida.



Pseudocódigo:

```
...
IZQ ← 2
DER ← N
iterar
  para J de DER a IZQ con incremento -1 hacer
    si V(J - 1) > V(J)
      entonces
        AUX ← V(J - 1)
        V(J - 1) ← V(J)
        V(J) ← AUX
    finsi
  finpara
  IZQ ← IZQ + 1
  para J de IZQ a DER hacer
    si V(J - 1) > V(J)
      entonces
        AUX ← V(J - 1)
        V(J - 1) ← V(J)
        V(J) ← AUX
    finsi
  finpara
  DER ← DER - 1
  salir si IZQ > DER
finiterar
...
```

Ordinograma:

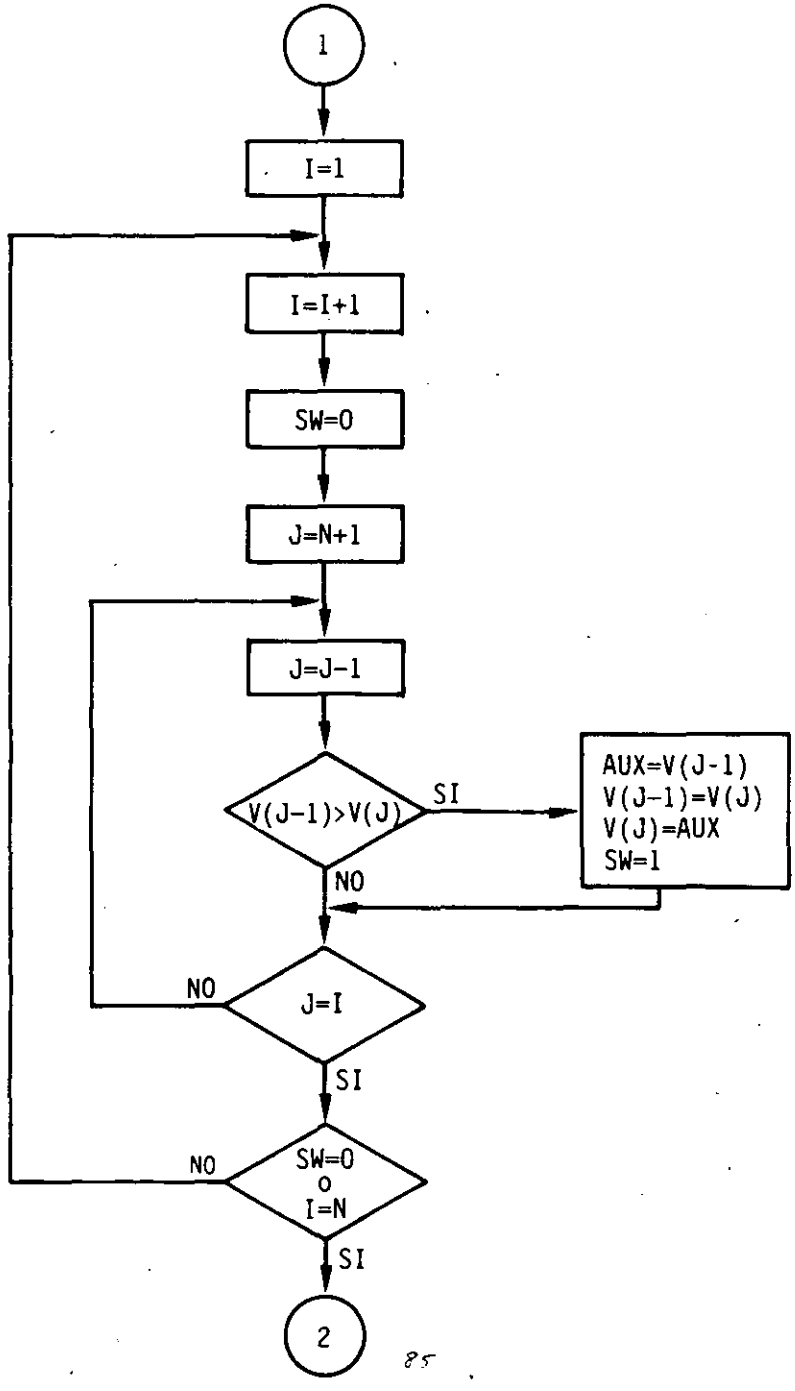


6.3.5. ORDENACION POR INTERCAMBIO DIRECTO CON TEST DE COMPROBACION

Es una mejora de los métodos de intercambio directo en la que se comprueba mediante un **switch** (interruptor) si el vector está totalmente ordenado después de cada pasada, terminando la ejecución en caso afirmativo.

Vamos a aplicar esta mejora al método de la burbuja con recorrido de derecha a izquierda.

Ordinograma:



Pseudocódigo:

```
...
I ← 1
iterar
  I ← I + 1
  SW ← 0
  para J de N a I con incremento -1 hacer
    si V(J - 1) > V(J)
       entonces
         AUX ← V(J - 1)
         V(J - 1) ← V(J)
         V(J) ← AUX
         SW ← 1
    finsi
  finpara
  salir si SW = 0 o I = N
finiterar
...
```

6.3.6. ORDENACION POR INSERCION CON INCREMENTOS DECRECIENTES. METODO SHELL

Es una mejora del método de inserción directa en la cuál se produce un acercamiento de los elementos descolocados hacia su posición correcta en saltos de mayor longitud.

Se repite un mismo proceso con una distancia de comparación que inicialmente es la mitad de la longitud del vector y que se va reduciendo a la mitad en cada repetición hasta que dicha distancia vale 1.

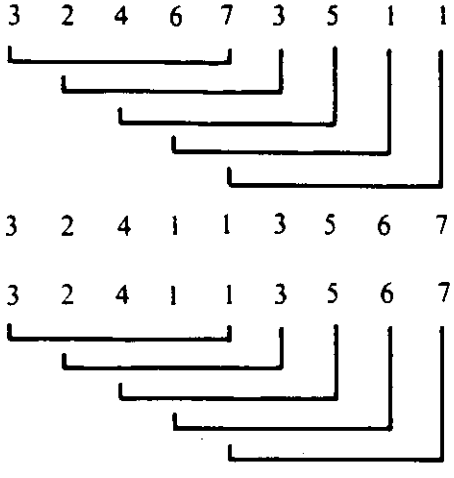
Cada pasada termina al detectarse mediante un switch la ordenación a la distancia correspondiente.

Ejemplo:

Secuencia inicial

3 2 4 6 7 3 5 1 1

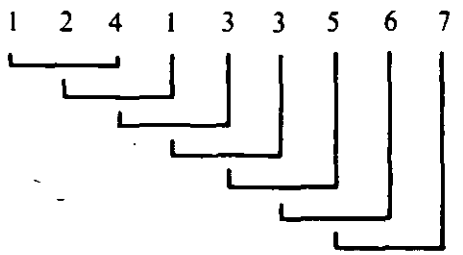
Primer paso. Distancia de comparación 4.



3 2 4 1 1 3 5 6 7

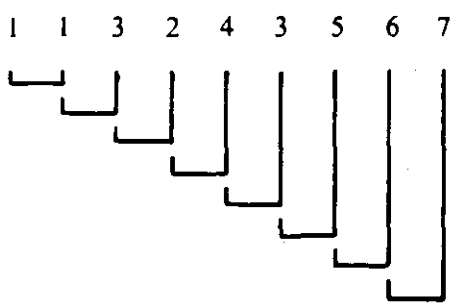
1 2 4 1 3 3 5 6 7 ordenados los pares de distancia 4.

Segundo paso. Distancia de comparación 2.



1 1 3 2 4 3 5 6 7 ordenados los pares de distancia 2.

Tercer paso. Distancia de comparación 1.

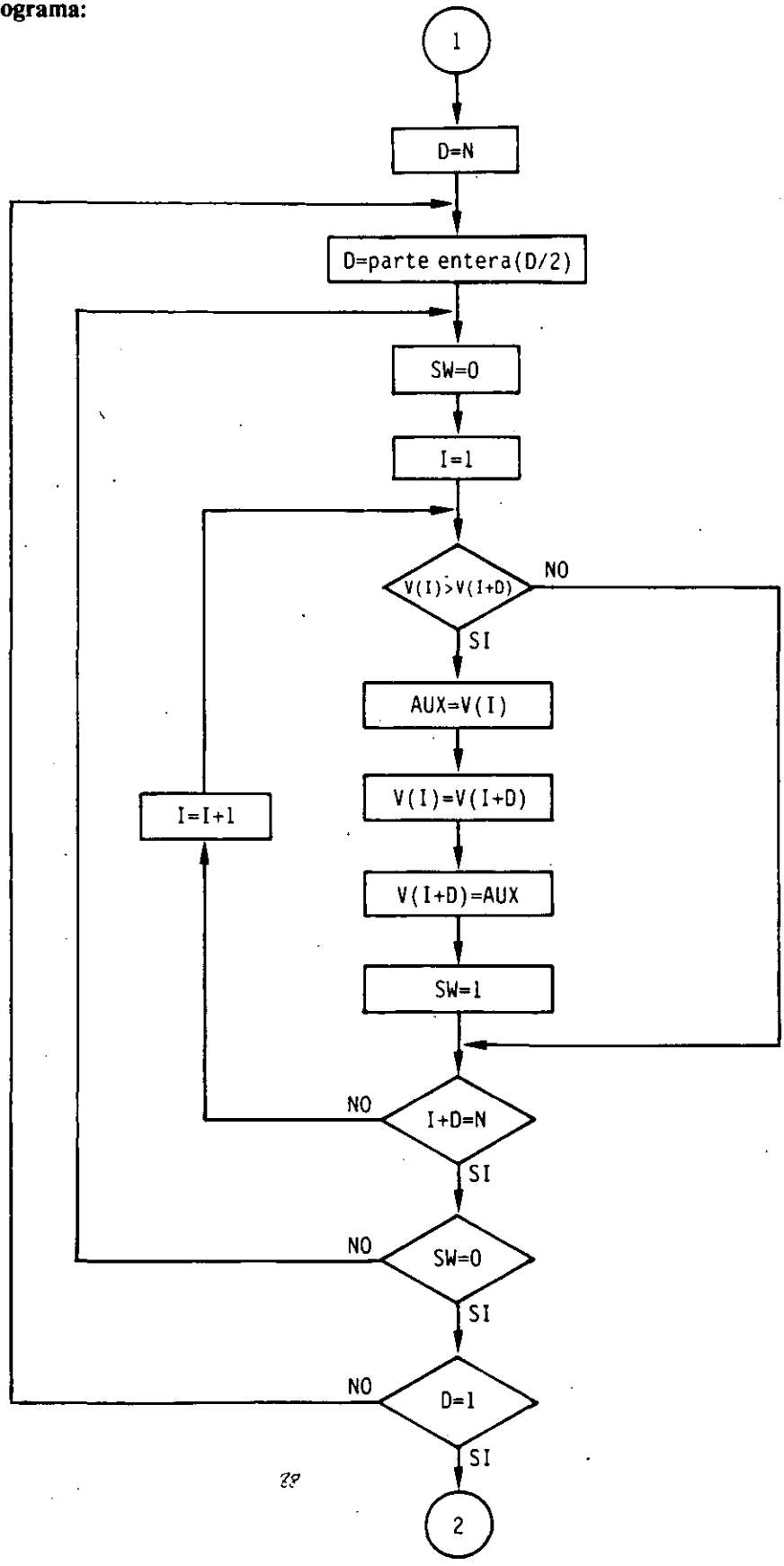


1 1 2 3 3 4 5 6 7 ordenados los pares de distancia 1.

Pseudocódigo:

```
...
D ← N
iterar
  D ← parte entera (D / 2)
  iterar
    SW ← 0
    I ← 1
    iterar
      si V(I) > V(I + D)
        entonces
          AUX ← V(I)
          V(I) ← V(I + D)
          V(I + D) ← AUX
           SW ← 1
      finsi
      salir si I + D = N
      I ← I + 1
    finiterar
    salir si SW = 0
  finiterar
  salir si D = 1
finiterar
...
```

Ordinograma:



Estructuras de datos externas (archivos)

7.0. INTRODUCCION

Los objetos tratados por un programa, que hemos visto hasta ahora, tienen dos limitaciones importantes: por un lado, la cantidad de datos que pueden almacenar es bastante reducida, por ser limitada la memoria central del computador; por otro, su existencia está condicionada al tiempo que dure la ejecución del programa; es decir, cuando termina el programa, todos sus datos desaparecen de la memoria central.

Para abordar un aspecto importante de la Programación, que trata de la manipulación y almacenamiento de datos para futuros usos, se utilizan las estructuras de datos externas denominadas ficheros o archivos.

Los archivos no están contenidos en la memoria central del computador, sino que residen en soportes externos, que abren comunicación con el mismo al ser solicitada.

Su nombre corresponde al concepto clásico de conjunto de fichas conteniendo información relativa a un mismo tema. Por ejemplo, el archivo de un hospital, que contiene los historiales clínicos de los enfermos, o el archivo de una biblioteca, que contiene información sobre los libros existentes en la misma.

Los soportes donde residen estos archivos pueden ser una carpeta, un armario, etc., existiendo algunas reglas o criterios de clasificación y manipulación.

Desde el punto de vista informático, un archivo es algo similar, residente en un soporte de información externo como puede ser un disco o una cinta magnética.

Un archivo o fichero se compone de registros (equivalentes a las fichas), siendo éstos la unidad de acceso y tratamiento.

Esta estructura es fundamental, debido a que nos permite almacenar cualquier tipo de información, como por ejemplo datos, textos, gráficos, programas, etc.

En este capítulo estudiaremos la creación y manejo de archivos de datos.

7.1. CONCEPTOS Y DEFINICIONES

Un **archivo o fichero** es un conjunto de informaciones estructuradas en unidades de acceso denominadas registros, todos del mismo tipo.

Un **registro** (registro lógico) es una estructura de datos formada por uno o más elementos denominados **campos**, que pueden ser de diferentes tipos y que, a su vez, pueden estar compuestos por subcampos.

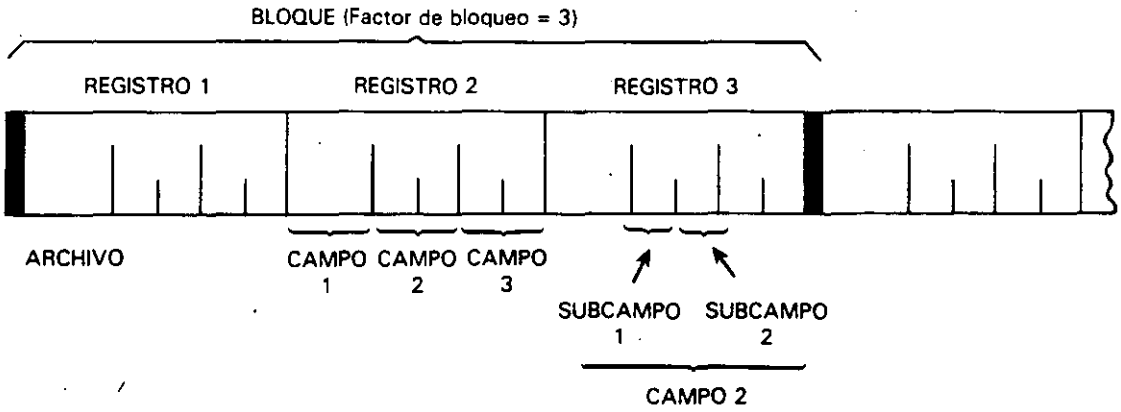
Los archivos contienen información relativa a un conjunto de individuos u objetos por regla general, estando ubicada la información correspondiente a cada uno de ellos en un registro.

Se denomina **clave** o identificativo a un campo especial del registro que sirve para identificarlo.

Algunos archivos en sus registros no tienen campo clave, mientras que otros pueden tener varios, denominándose éstos, clave primaria, secundaria, etc.

Otro concepto es el de **bloque** (registro físico), correspondiente a la cantidad de información que se transfiere en cada operación de lectura o escritura sobre un archivo. Su tamaño depende de las características físicas del computador utilizado.

Se denomina **factor de bloqueo** al número de registros lógicos que contiene cada bloque.



7.2. CARACTERISTICAS DE LOS ARCHIVOS

Las principales características de esta estructura son:

— Independencia de las informaciones respecto de los programas.
— Un archivo puede ser accedido por distintos programas en distintos momentos.
— La información almacenada es permanente.
— Gran capacidad de almacenamiento.

7.3. CLASIFICACION DE LOS ARCHIVOS SEGUN SU USO

Los archivos se clasifican según su uso en tres grupos:

— **Permanentes o maestros.** Contienen información que varía poco. En algunos casos es preciso actualizarlos periódicamente.
— **De movimientos.** Se crean para actualizar los archivos maestros. Sus registros son de tres tipos: altas, bajas y modificaciones.
— **De maniobra o trabajo.** Tienen una vida limitada, normalmente menor que la duración de la ejecución de un programa. Se utilizan como auxiliares de los anteriores.

7.4. ORGANIZACION DE ARCHIVOS

Los archivos se organizan para su almacenamiento y acceso, según las necesidades de las aplicaciones que los van a utiliar.

Limitamos el estudio a las organizaciones más utilizadas, que son:

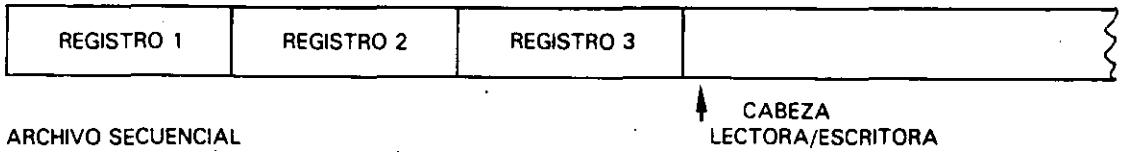
— **Secuencial.**
— **Directa o aleatoria.**
— **Secuencial indexada.**

7.4.1. ORGANIZACION SECUENCIAL

Es aquella en la cual los registros ocupan posiciones consecutivas de memoria y que sólo se puede acceder a ellos de uno en uno a partir del primero.

En un archivo secuencial, no se pueden hacer operaciones de escritura cuando se está leyendo, ni operaciones de lectura cuando se está escribiendo.

Por otro lado, para actualizarlos es preciso crear nuevos archivos donde se copien los antiguos junto con las actualizaciones.



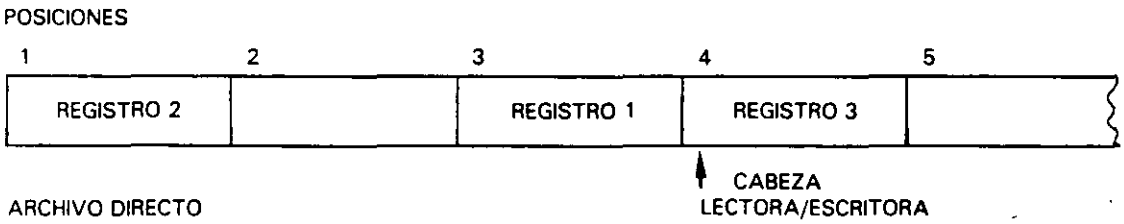
7.4.2. ORGANIZACION DIRECTA O ALEATORIA

Las informaciones se colocan y se acceden aleatoriamente mediante su posición, es decir, indicando el lugar relativo que ocupan dentro del conjunto de posiciones posibles.

En esta organización se pueden leer y escribir registros, en cualquier orden y en cualquier lugar.

Presenta el inconveniente de que es tarea del programador establecer la relación entre la posición que ocupa un registro y su contenido, además puede desaprovecharse parte del espacio destinado al archivo, ya que pueden quedar huecos libres entre unos registros y otros.

Su principal ventaja es la rapidez de acceso a un registro cualquiera, ya que para ello no es preciso pasar por los anteriores.



7.4.3. ORGANIZACION SECUENCIAL INDEXADA

Un archivo con esta organización consta de tres áreas:

— Area de índices.
— Area primaria.
— Area de excedentes (*overflow*).

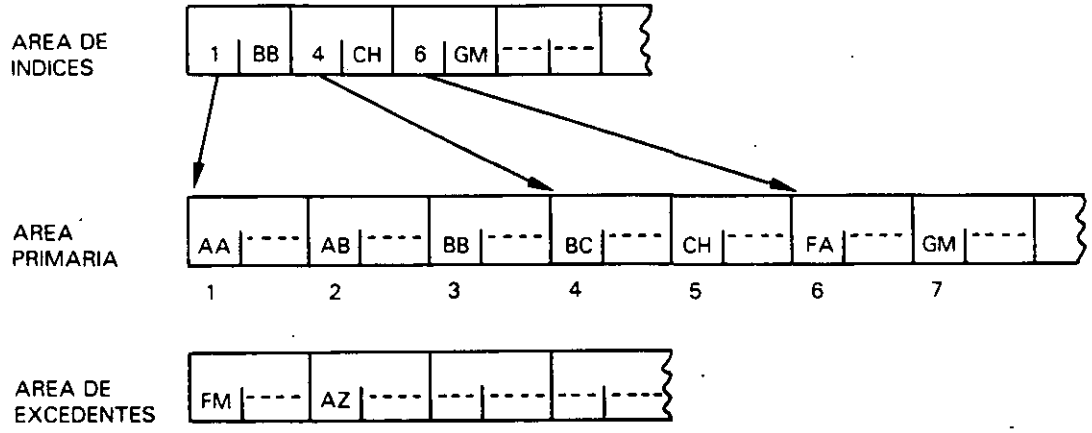
El área primaria contendrá los registros de datos, clasificados en orden ascendente por su campo clave.

El área de índices es un archivo secuencial creado por el sistema, en el que cada registro establece una división (segmento) en el área primaria, y contiene la dirección de comienzo del segmento y la clave más alta del mismo. De esta manera el sistema accede de forma directa a un segmento del área primaria a partir del área de índices, de forma similar a la búsqueda de un capítulo de un libro a partir de su índice.

Por último, se reserva un espacio, llamado área de excedentes, para añadir nuevos registros que no pueden ser colocados en el área primaria cuando se produce una actualización del archivo.

Esta organización presenta la ventaja de un rápido acceso y además, el sistema se encarga de relacionar la posición de cada registro con su contenido por medio del área de índices. También es trabajo del sistema la gestión de las áreas de índices y excedentes.

Los inconvenientes que presenta son la necesidad de espacio adicional para el área de índices y el desaprovechamiento de espacio que resulta de quedar huecos intermedios libres después de sucesivas actualizaciones.



7.5. OPERACIONES SOBRE ARCHIVOS

Las operaciones generales que se realizan sobre un archivo son:

— **Creación.** Escritura de todos sus registros.
— **Consulta.** Lectura de todos sus registros.
— **Actualización.** Inserción, supresión o modificación de algunos de sus registros.
— **Clasificación.** Reubicación de los registros de tal forma que queden ordenados según determinados criterios.
— **Borrado.** Eliminación total del archivo, dejando libre el espacio del soporte que ocupaba.

Las operaciones más usuales a nivel de registro son:

— **Inserción.** Añadir un nuevo registro al archivo.
— **Supresión.** Quitar un registro del archivo.
— **Modificación.** Alterar la información de un registro.
— **Consulta.** Leer el contenido de un registro.

7.6. INSTRUCCIONES PARA MANEJO DE ARCHIVOS

Utilizamos como ejemplo para el estudio de las instrucciones, un archivo denominado AGENDA, cuyos registros contienen 4 campos con el NOMBRE, EDAD, DIRECCION y TELEFONO de una serie de personas.

AGENDA

RUBIO SANZ, PABLO	19	RELOJ, 5 28013 MADRID	273 46 00
NOMBRE	EDAD	DIRECCION	TELEFONO

PERSONA

En primer lugar, es preciso declarar el archivo, su nombre y la estructura de sus registros.

Pseudocódigo:

```
AGENDA es archivo de PERSONA
PERSONA es registro compuesto de
          NOMBRE es alfanumérico
          EDAD es numérico entero
          DIRECCION es alfanumérico
          TELEFONO es alfanumérico
        finregistro
```

BASIC: No se declaran previamente, sino que se hace en la instrucción de apertura del archivo.

COBOL: Dentro de la FILE SECTION de la DATA DIVISION.

```
FD  AGENDA LABEL RECORD  STANDARD VALUE OF FILE-ID 'AGENDA.DAT'.
01  PERSONA.
    02 NOMBRE    PIC X(20).
    02 EDAD      PIC 99.
    02 DIRECCION PIC X(30).
    02 TELEFONO  PIC X(12).
```

El archivo lógico AGENDA se asigna a un dispositivo físico en la ENVIRONMENT DIVISION:

```
SELECT AGENDA ASSIGN TO DISK.
```

Pascal:

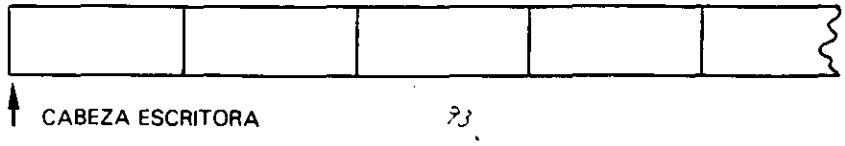
```
TYPE RPERSONA = RECORD
                  NOMBRE    : STRING[20];
                  EDAD      : INTEGER;
                  DIRECCION : STRING[30];
                  TELEFONO  : STRING[12]
                END;
     FAGENDA=FILE OF RPERSONA;
VAR AGENDA  : FAGENDA;
    PERSONA : RPERSONA;
```

El archivo lógico AGENDA se asigna a un dispositivo físico con la siguiente instrucción, que se debe ejecutar antes de la apertura del mismo:

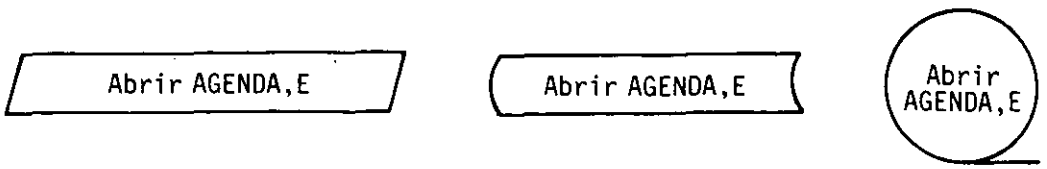
```
ASSIGN(AGENDA, 'AGENDA.DAT');
```

7.6.1. CREACION DE ARCHIVOS SECUENCIALES

• **Apertura:** Reserva un archivo secuencial en exclusividad para el programa que la ejecuta, coloca la cabeza de escritura en su primer registro, quedando preparado para ser creado.
Si el archivo no existía, lo crea y si ya existía, borra todo su contenido anterior.



Ordinograma:



Pseudocódigo:

```
abrir AGENDA para escritura
```

BASIC: Hay dos instrucciones equivalentes.

```
OPEN "O", #1, "AGENDA.DAT"
OPEN "AGENDA.DAT" FOR OUTPUT AS #1
```

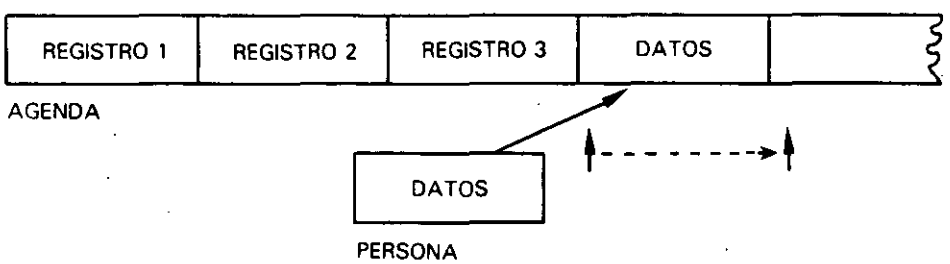
COBOL:

```
OPEN OUTPUT AGENDA.
```

Pascal:

```
REWRITE(AGENDA);
```

• **Escritura de un registro:** Coloca en el lugar donde esté la cabeza de escritura el contenido del registro *buffer*, al que previamente se le habrán asignado los datos, avanzando la cabeza de escritura al siguiente registro del archivo.



Ordinograma:

Pseudocódigo:

```
escribir AGENDA, PERSONA
```

BASIC: Se utilizan dos instrucciones para grabar los datos contenidos en determinadas variables del programa.

```
PRINT #1, NOMBRE$, EDAD, DIRECCION$, TELEFONO$
WRITE #1, NOMBRE$, EDAD, DIRECCION$, TELEFONO$
```

COBOL:

```
WRITE PERSONA.
```

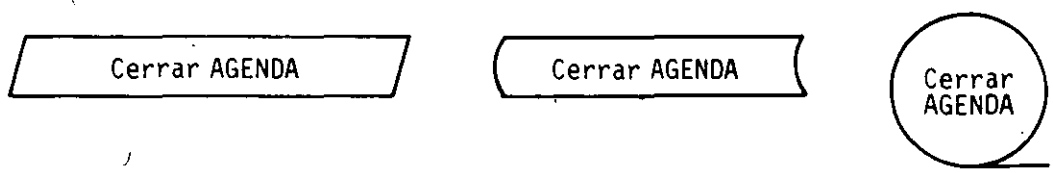
Pascal:

```
WRITE(AGENDA, PERSONA);
```

• **Cierre:** Libera el archivo AGENDA del programa, coloca una marca especial de "fin de archivo" que será detectada en posteriores lecturas. El archivo queda a disposición de cualquier programa que lo solicite.

AGENDA

Ordinograma:



Pseudocódigo:

```
cerrar AGENDA
```

BASIC:

```
CLOSE #1
```

COBOL:

```
CLOSE AGENDA.
```

Pascal:

```
CLOSE(AGENDA);
```

Ejercicio:

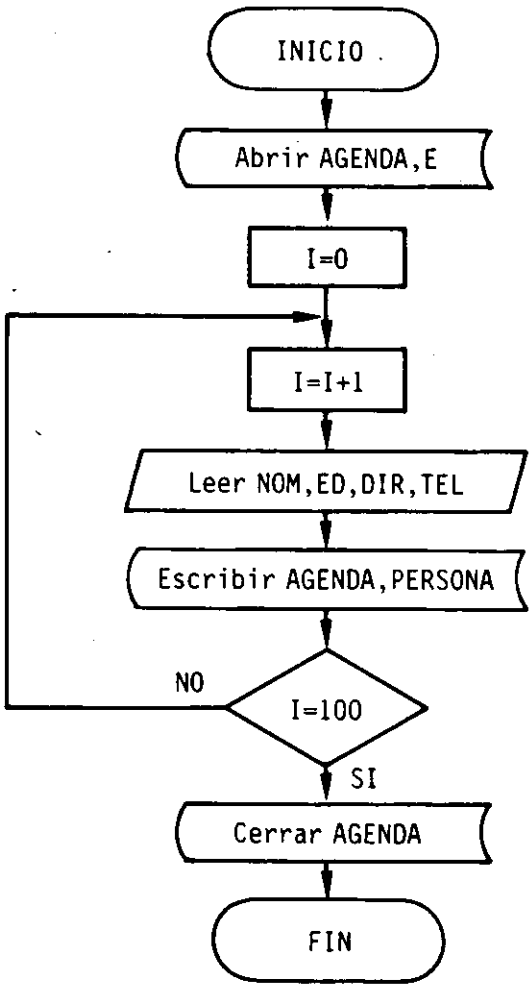
Creación del archivo AGENDA en disco, con los datos de 100 personas, introducidos por teclado.

Pseudocódigo:

```
Programa CREACION AGENDA
Entorno:
  AGENDA es archivo de PERSONA
  PERSONA es registro compuesto de
              NOMBRE es alfanumérico
              EDAD es numérico entero
              DIRECCION es alfanumérico
```

```
                TELEFONO es alfanumérico
              finregistro
  I es numérica entera
Algoritmo:
  abrir AGENDA para escritura
  para I de 1 a 100 hacer
    leer NOMBRE, EDAD, DIRECCION, TELEFONO
    escribir AGENDA, PERSONA
  finpara
  cerrar AGENDA
Finprograma
```

Ordinograma:



Codificación BASIC:

```
10 REM CREACION AGENDA
20 CLS
30 OPEN "AGENDA.DAT" FOR OUTPUT AS #1
40 FOR I = 1 TO 100
50   CLS
60   PRINT "Datos de la persona n° "; I
```

```
70   INPUT "Nombre "; NOMBRE$
80   INPUT "Edad "; EDAD
90   INPUT "Dirección "; DIRECCION$
100  INPUT "Teléfono "; TELEFONO$
110  WRITE #1, NOMBRE$, EDAD, DIRECCION$, TELEFONO$
120 NEXT I
130 CLOSE #1
140 END
```

Codificación COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CREACION-AGENDA.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT AGENDA ASSIGN TO DISK.
DATA DIVISION.
FILE SECTION.
FD  AGENDA LABEL RECORD STANDARD VALUE OF FILE-ID 'AGENDA.DAT'.
01  PERSONA.
    02 NOMBRE    PIC X(20).
    02 EDAD      PIC 99.
    02 DIRECCION PIC X(30).
    02 TELEFONO  PIC X(12).
WORKING-STORAGE SECTION.
01  AUX-PERSONA.
    02 AUX-NOMBRE    PIC X(20).
    02 AUX-EDAD      PIC 99.
    02 AUX-DIRECCION.PIC X(30).
    02 AUX-TELEFONO  PIC X(12).
77  I PIC 999.
77  S PIC ZZ9.
PROCEDURE DIVISION.
INICIO.
    OPEN OUTPUT AGENDA.
    PERFORM CREACION VARYING I FROM 1 BY 1 UNTIL I > 100.
FIN.
    CLOSE AGENDA.
STOP RUN.
CREACION.
    MOVE I TO S.
    DISPLAY 'Datos de la persona n° ', S.
    DISPLAY 'Nombre: '.
    ACCEPT AUX-NOMBRE.
    DISPLAY 'Edad: '.
    ACCEPT AUX-EDAD.
    DISPLAY 'Dirección: '.
    ACCEPT AUX-DIRECCION.
    DISPLAY 'Telefono: '.
    ACCEPT AUX-TELEFONO.
    WRITE PERSONA FROM AUX-PERSONA.
```

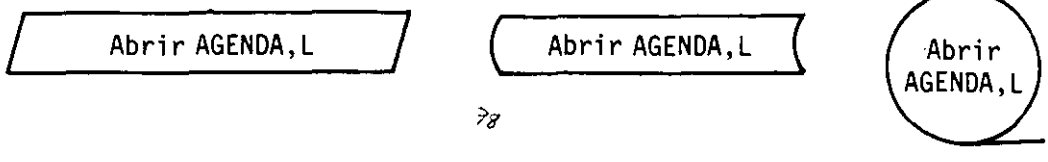
Codificación Pascal:

```
PROGRAM CREACIONAGENDA (INPUT, OUTPUT, AGENDA);
TYPE RPERSONA = RECORD
                  NOMBRE    : STRING[20];
                  EDAD      : INTEGER;
                  DIRECCION : STRING[30];
                  TELEFONO  : STRING[12]
                END;
     FAGENDA = FILE OF RPERSONA;
VAR AGENDA  : FAGENDA;
    PERSONA : RPERSONA;
    I : INTEGER;
BEGIN (*PP*)
  ASSIGN(AGENDA, 'AGENDA.DAT');
  REWRITE(AGENDA);
  FOR I := 1 TO 100 DO
    BEGIN (*1*)
      CLRSCR;
      WRITELN('Datos de la persona n° ', I);
      WITH PERSONA DO
        BEGIN (*2*)
          WRITE('Nombre: ');
          READLN(NOMBRE);
          WRITE('Edad: ');
          READLN(EDAD);
          WRITE('Dirección: ');
          READLN(DIRECCION);
          WRITE('Teléfono: ');
          READLN(TELEFONO)
        END; (*2*)
      WRITE(AGENDA, PERSONA)
    END; (*1*)
  CLOSE(AGENDA)
END. (*PP*)
```

7.6.2. LECTURA DE ARCHIVOS SECUENCIALES

• **Apertura:** Reserva un archivo secuencial en exclusividad para un programa. Coloca la cabeza lectora sobre el primer registro, quedando el archivo preparado para ser leído.

Ordinograma:



Pseudocódigo:

```
abrir AGENDA para lectura
```

BASIC: Se expresa indistintamente de dos formas.

```
OPEN "I", #1, "AGENDA.DAT"
OPEN "AGENDA.DAT" FOR INPUT AS #1
```

COBOL:

```
OPEN INPUT AGENDA.
```

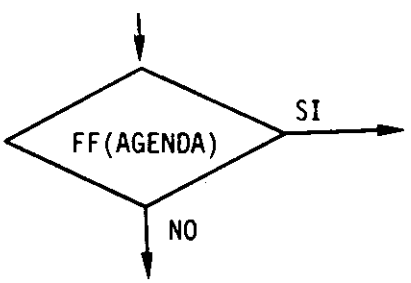
Pascal:

```
RESET(AGENDA);
```

• **Comprobación de final de archivo:** Es una función booleana que toma el valor CIERTO si la cabeza lectora señala la marca especial de fin de archivo, es decir, si no quedan registros por leer, y toma el valor FALSO en caso contrario.

Siempre habrá que hacer esta comprobación antes de leer un nuevo registro, pues si se ejecuta una instrucción de lectura cuando no quedan registros por leer se producirá un error y se interrumpirá la ejecución del programa.

Ordinograma:



Pseudocódigo:

```
ff(AGENDA)
```

BASIC:

```
EOF(1)
```

COBOL:

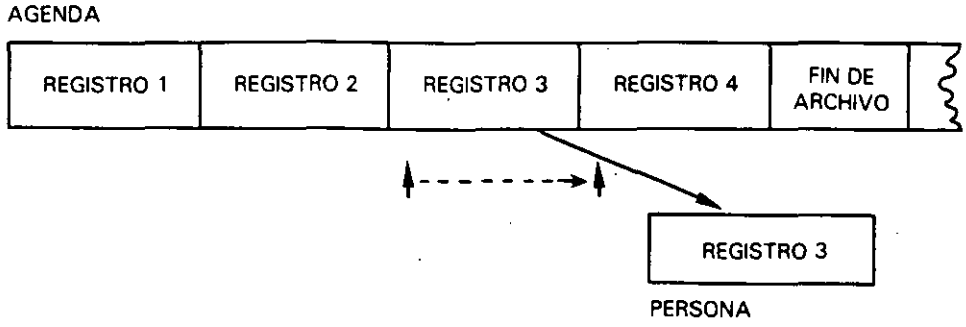
```
AT END
```

(Se escribe acompañando a la instrución de lectura.)

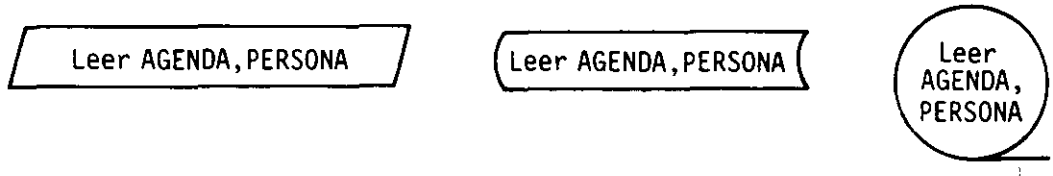
Pascal:

```
EOF(AGENDA)
```

• **Lectura de un registro:** Coloca el contenido del registro apuntado por la cabeza lectora sobre la variable *buffer* del archivo, avanzando la cabeza lectora al siguiente registro.



Ordinograma:



Pseudocódigo:

```
leer AGENDA, PERSONA
```

BASIC:

```
INPUT #1, NOMBRE$, EDAD, DIRECCION$, TELEFONO$
```

COBOL:

```
READ AGENDA.
```

Pascal:

```
READ(AGENDA, PERSONA);
```

• **Cierre:** Libera el archivo del programa. El archivo queda igual que estaba antes de ser leído, sin ninguna modificación, tanto si se ha leído completo como si no. Asimismo, queda a disposición de cualquier programa que lo solicite.

La notación es igual que para el caso de escritura.

Ejercicio:

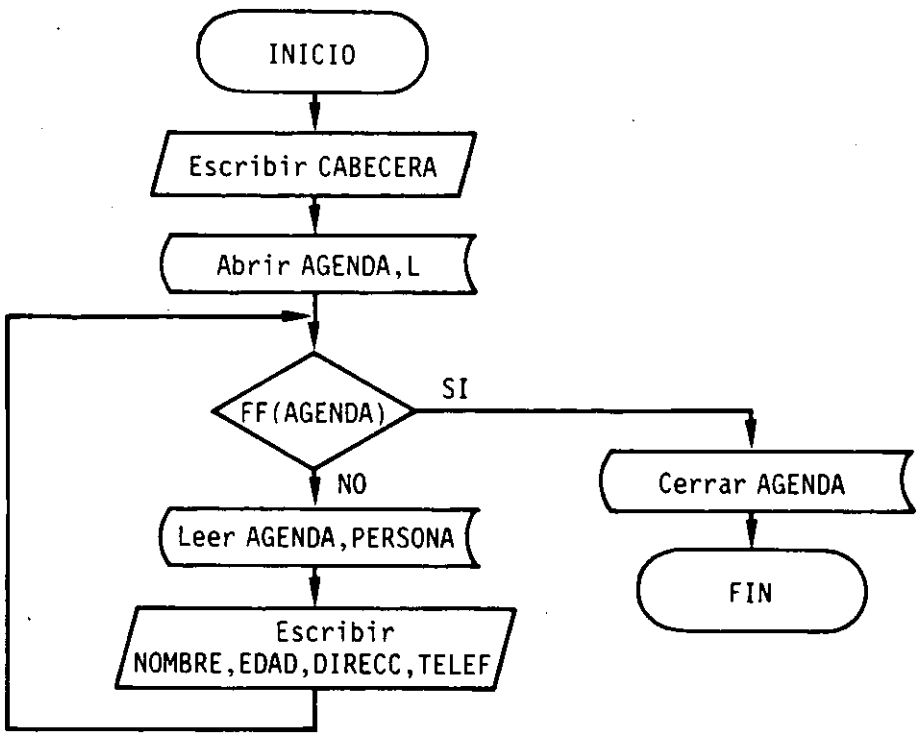
Listado por impresora del archivo AGENDA del ejemplo anterior.

Pseudocódigo:

```
Programa LISTADO AGENDA
Entorno:
  AGENDA es archivo de PERSONA
  PERSONA es registro compuesto de
              NOMBRE es alfanumérico
              EDAD es numérico entero
              DIRECCION es alfanumérico
              TELEFONO es alfanumérico
            finregistro
```

```
Algoritmo:
  escriba cabecera
  abrir AGENDA para lectura
  mientras no ff(AGENDA) hacer
    leer AGENDA, PERSONA
    escribir NOMBRE, EDAD, DIRECCION, TELEFONO
  finmientras
  cerrar AGENDA
Finprograma
```

Ordinograma:



Codificación BASIC:

```
10 REM LISTADO AGENDA
20 LPRINT "LISTADO DE LA AGENDA"
30 LPRINT "Nombre", "Edad", "Dirección", "Teléfono"
40 OPEN "AGENDA.DAT" FOR INPUT AS #1
50 WHILE NOT EOF(1)
60   INPUT #1, NOMBRE$, EDAD, DIRECCION$, TELEFONO$
70   LPRINT NOMBRE$, EDAD, DIRECCION$, TELEFONO$
80 WEND
90 CLOSE #1
100 END
```

Codificación COBOL:

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. LISTADO-AGENDA.
 ENVIRONMENT DIVISION.
```

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT AGENDA ASSIGN TO DISK.
    SELECT LISTADO ASSIGN TO PRINTER.
DATA DIVISION.
FILE SECTION.
FD  AGENDA LABEL RECORD STANDARD VALUE OF FILE-ID 'AGENDA.DAT'.
01  PERSONA.
    02 NOMBRE    PIC X(20).
    02 EDAD      PIC 99.
    02 DIRECCION PIC X(30).
    02 TELEFONO  PIC X(12).
FD  LISTADO LABEL RECORD OMITTED.
01  LINEA PIC X(80).
WORKING-STORAGE SECTION.
01  LIN-PERSONA.
    02 NOMBRE    PIC X(20).
    02 FILLER    PIC X(4) VALUE SPACES.
    02 EDAD      PIC 99.
    02 FILLER    PIC X(4) VALUE SPACES.
    02 DIRECCION PIC X(30).
    02 FILLER    PIC X(4) VALUE SPACES.
    02 TELEFONO  PIC X(12).
    02 FILLER    PIC X(4) VALUE SPACES.
01  CABECERA1.
    02 FILLER PIC X(20) VALUE 'LISTADO DE LA AGENDA'.
    02 FILLER PIC X(60) VALUE SPACES.
01  CABECERA2.
    02 FILLER    PIC X(6) VALUE 'Nombre'.
    02 FILLER    PIC X(18) VALUE SPACES.
    02 FILLER    PIC X(4) VALUE 'Edad'.
    02 FILLER    PIC X(2) VALUE SPACES.
    02 FILLER    PIC X(9) VALUE 'Dirección'.
    02 FILLER    PIC X(25) VALUE SPACES.
    02 FILLER    PIC X(8) VALUE 'Teléfono'.
    02 FILLER    PIC X(8) VALUE SPACES.
PROCEDURE DIVISION.
INICIO.
    OPEN INPUT AGENDA, OUTPUT LISTADO.
    WRITE LINEA FROM CABECERA1.
    WRITE LINEA FROM CABECERA2.
LECTURA.
    READ AGENDA AT END GO TO FIN.
    MOVE CORRESPONDING PERSONA TO LIN-PERSONA.
* Si la versión utilizada no dispone de la opción
* CORRESPONDING, habrá que asignar uno a uno los
* campos de PERSONA a los de LIN-PERSONA.
     WRITE LINEA FROM LIN-PERSONA.
     GO TO LECTURA.
 FIN.
     CLOSE AGENDA, LISTADO.
     STOP RUN.
```

Codificación Pascal:

```
PROGRAM LISTADOAGENDA (AGENDA, LST);
TYPE RPERSONA = RECORD
                  NOMBRE    : STRING[20];
                  EDAD      : INTEGER;
                  DIRECCION : STRING[30];
                  TELEFONO  : STRING[12]
                END;
     FAGENDA = FILE OF RPERSONA;
VAR AGENDA  : FAGENDA;
    PERSONA : RPERSONA;
BEGIN (*PP*)
  ASSIGN(AGENDA, 'AGENDA.DAT');
  WRITELN(LST, 'LISTADO DE LA AGENDA');
  WRITELN(LST, 'Nombre', ' ':18, 'Edad', '  ', 'Dirección', ' ':25,
               'Teléfono');
  RESET(AGENDA);
  WHILE NOT EOF(AGENDA) DO
    BEGIN (*1*)
      READ(AGENDA, PERSONA);
      WITH PERSONA DO
        WRITELN(LST, NOMBRE, EDAD, DIRECCION, TELEFONO)
    END; (*1*)
  CLOSE(AGENDA)
END. (*PP*)
```

7.6.3. LECTURA-ESCRITURA DE ARCHIVOS DIRECTOS

• **Apertura:** Reserva un archivo directo en exclusividad para un programa. Si no existía anteriormente, lo crea. El archivo queda preparado para realizar operaciones de lectura o escritura sobre el mismo.

Ordinograma:

Pseudocódigo:

```
abrir AGENDA directo
```

BASIC: Para esta organización es preciso especificar la longitud del registro (por defecto, se asume una longitud de 128 caracteres).

```
OPEN "R", #1, "AGENDA", 64
OPEN "AGENDA" AS #1 LEN=64
```

Es obligatorio declarar la estructura del registro cada vez que se abre un archivo, con la siguiente instrución declarativa:

```
FIELD #1, 20 AS NOMBRE$, 2 AS EDAD$, 30 AS DIRECCION$,
      12 AS TELEFONO$
```

Todos los campos han de ser obligatoriamente alfanuméricos, por lo que será necesario transformar los datos numéricos en alfanuméricos mediante las funciones predefinidas:

```
STR$, MKI$, MKS$ y MKD$.
```

Para asignar valores a los campos del registro se utilizan las instrucciones LSET y RSET que ajustan el dato a la izquierda y a la derecha respectivamente.

```
RSET EDAD$ = MKI$(EDAD%)
```

Donde EDAD% es la variable entera que contiene el dato a almacenar.

Cuando se extraen los datos del archivo directo, los correspondientes a valores numéricos hay que transformarlos de nuevo, para poder operar con ellos. Para ello se utilizan las funciones predefinidas:

```
VAL, CVI, CVS y CVD.

EDAD%=CVI(EDAD$)
```

COBOL: Previamente a la apertura, se habrá declarado en la sentencia SELECT un dispositivo de acceso directo, el tipo de acceso y la clave.

```
SELECT AGENDA ASSIGN TO DISK
              ORGANIZATION IS RELATIVE
              ACCESS IS RANDOM
              RELATIVE KEY IS NUM.
```

La clave declarada NUM es un campo numérico declarado en la WORKING-STORAGE SECTION.

```
OPEN I-O AGENDA.
```

Pascal: Se abre igual que para lectura, permitiendo operaciones de lectura y escritura.

```
RESET(AGENDA);
```

• **Escritura de un registro:** Coloca en una posición cualquiera del archivo, fijada en el programa, el contenido del registro *buffer*, al que previamente se le habrán asignado los datos.

Ordinograma:

Pseudocódigo:

```
escribir AGENDA, PERSONA, NUM
```

BASIC:

```
PUT #1, NUM%
```

COBOL:

```
WRITE PERSONA.
```

Pascal:

```
SEEK(AGENDA, NUM);
WRITE(AGENDA, PERSONA);
```

• **Lectura de un registro:** Coloca el contenido del registro cuya posición se indique, en el registro *buffer*.

Ordinograma:

Pseudocódigo:

```
leer AGENDA, PERSONA, NUM
```

BASIC:

```
GET #1, NUM%
```

COBOL:

```
READ AGENDA.
```

Pascal:

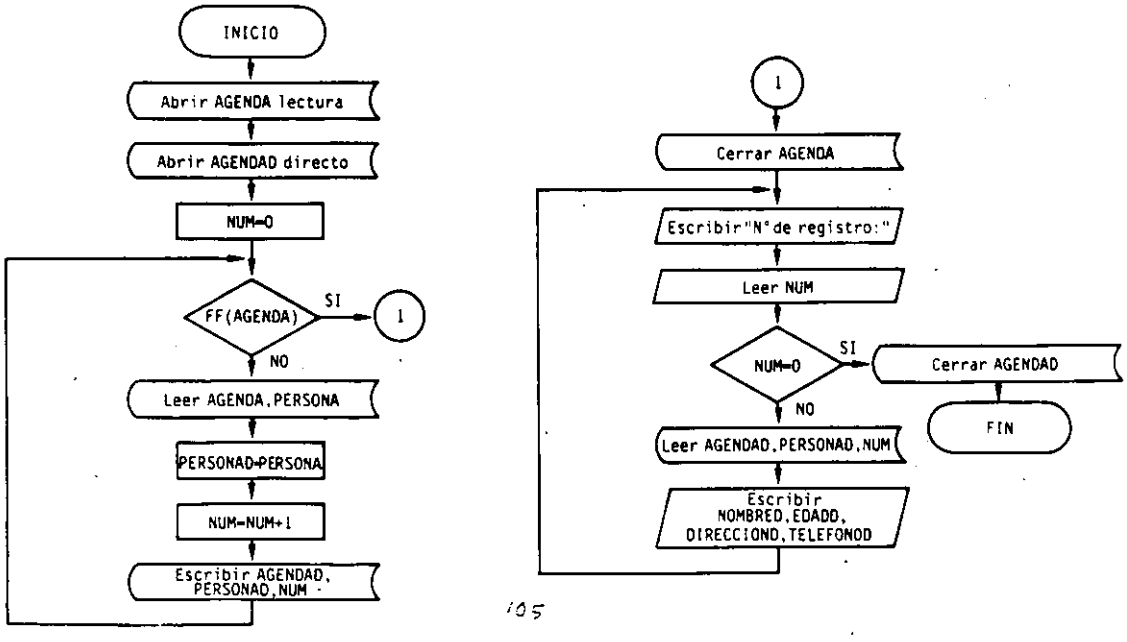
```
SEEK(AGENDA, NUM);
READ(AGENDA, PERSONA);
```

• **Cierre:** Igual que en los casos anteriores.

Ejercicio:

Programa que copia el archivo secuencial AGENDA del ejemplo anterior en un archivo directo AGENDAD, conservando cada registro su posición relativa. Concluida la copia, el programa permitirá consultas al archivo directo, introduciendo por teclado el número de registro, y terminando al introducir un 0.

Ordinograma:



Pseudocódigo:

```
Programa AGENDA DIRECTA
Entorno:
  AGENDA es archivo  de PERSONA
  PERSONA es registro compuesto de
               NOMBRE es alfanumérico
               EDAD es numérico entero
               DIRECCION es alfanumérico
               TELEFONO es alfanumérico
             finregistro
  AGENDAD es archivo de PERSONAD
  PERSONAD es registro compuesto de
               NOMBRED es alfanumérico
               EDADD es numérico entero
               DIRECCIOND es alfanumérico
               TELEFONOD es alfanumérico
             finregistro
  NUM es numérica entera
Algoritmo:
  abrir AGENDA para lectura
  abrir AGENDAD directo
  ** Copia de AGENDA en AGENDAD
  NUM ← 0
  mientras no ff(AGENDA) hacer
    leer AGENDA, PERSONA
    PERSONAD ← PERSONA
    NUM ← NUM + 1
    escribir AGENDAD, PERSONAD, NUM
  finmientas
  cerrar AGENDA
  ** AGENDAD contendrá 100 registros
  ** Consultas de AGENDAD
  iterar
    escribir "Número de registro (entre 1 y 100), para terminar 0"
    leer NUM
    salir si NUM = 0
    leer AGENDAD, PERSONAD, NUM
    escribir NOMBRED, EDADD, DIRECCIOND, TELEFONOD
  finiterar
  cerrar AGENDAD
Finprograma
```

Codificación BASIC:

```
10 REM AGENDA DIRECTA
20 OPEN "AGENDA" FOR INPUT AS #1
30 OPEN "AGENDAD" AS #2 LEN = 64
40 FIELD #2, 20 AS NOMBRE$, 2 AS EDAD$, 30 AS DIRECCION$,
           12 AS TELEFONO$
50 REM Copia de AGENDA en AGENDAD
60 NUM% = 0
70 WHILE NOT EOF(1)
80   INPUT #1, NOM$, ED%, DIR$, TEL$
90   LSET NOMBRE$ = NOM$
```

```
100  LSET EDAD$ = MKI$(ED%)
110  LSET DIRECCION$ = DIR$
120  LSET TELEFONO$ = TEL$
130  NUM% = NUM% + 1
140  PUT #2, NUM%
150 WEND
160 CLOSE #1
170 REM Consultas de AGENDAD
180 CLS
190 PRINT "Número de registro (entre 1 y 100), para terminar 0"
200 INPUT NUM%
210 IF NUM% = 0 THEN GOTO 290
220 GET #2, NUM%
230 NOM$ = NOMBRE$
240 ED% = CVI(EDAD$)
250 DIR$ = DIRECCION$
260 TEL$ = TELEFONO$
270 PRINT NOM$; "  "; ED%; "  "; DIR$; "  "; TEL$
280 GOTO 190
290 CLOSE #2
300 END
```

Codificación COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. AGENDA-DIRECTA.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT AGENDA ASSIGN TO DISK.
    SELECT AGENDAD ASSIGN TO DISK
                   ORGANIZATION IS RELATIVE
                   ACCESS IS RANDOM
                   RELATIVE KEY IS NUM.
DATA DIVISION.
FILE SECTION.
FD  AGENDA LABEL RECORD STANDARD.
01  PERSONA.
    02 NOMBRE    PIC X(20).
    02 EDAD      PIC 99.
    02 DIRECCION PIC X(30).
    02 TELEFONO  PIC X(12).
FD  AGENDAD LABEL RECORD STANDARD.
01  PERSONAD.
    02 NOMBRED    PIC X(20).
    02 EDADD      PIC 99.
    02 DIRECCIOND PIC X(30).
    02 TELEFONOD  PIC X(12).
WORKING-STORAGE SECTION.
77  NUM PIC 999.
01  PERSONAS.
    02 NOMBRED    PIC X(20).
    02 FILLER     PIC X(2) VALUE SPACES.
    02 EDADD      PIC Z9.
```

```
           02 FILLER      PIC X(2) VALUE SPACES.
           02 DIRECCIOND PIC X(30).
           02 FILLER      PIC X(2) VALUE SPACES.
           02 TELEFONOD   PIC X(12).
       PROCEDURE DIVISION.
       INICIO.
           OPEN INPUT AGENDA I-O AGENDAD.
      * Copia de AGENDA en AGENDAD.
           MOVE 0 TO NUM.
       BUCLE-COPIA.
           READ AGENDA AT END GO TO FIN-COPIA.
           ADD 1 TO NUM.
           WRITE PERSONAD FROM PERSONA.
           GO TO BUCLE-COPIA.
       FIN-COPIA.
           CLOSE AGENDA.
      * Consultas de AGENDAD.
       BUCLE-CONSULTAS.
           DISPLAY 'Número de registro (entre 1 y 100), para terminar 0'.
           ACCEPT NUM.
           IF NUM = 0 GO TO FIN-CONSULTAS.
           READ AGENDAD.
           MOVE CORRESPONDING PERSONAD TO PERSONAS.
           DISPLAY PERSONAS.
           GO TO BUCLE-CONSULTAS.
       FIN-CONSULTAS.
           CLOSE AGENDAD.
           STOP RUN.
```

Codificación Pascal:

```
PROGRAM AGENDADIRECTA (INPUT, OUTPUT, AGENDA, AGENDAD);
TYPE RPERSONA = RECORD
                  NOMBRE    : STRING[20];
                  EDAD      : INTEGER;
                  DIRECCION : STRING[30];
                  TELEFONO  : STRING[12]
                END;
     FAGENDA = FILE OF RPERSONA;
VAR AGENDA, AGENDAD : FAGENDA;
    PERSONA : RPERSONA;
    NUM : INTEGER;
BEGIN (*PP*)
  ASSIGN(AGENDA, 'AGENDA.DAT');
  ASSIGN(AGENDAD, 'AGENDAD.DAT');
  RESET(AGENDA);
  RESET(AGENDAD);
  (* Copia de AGENDA en AGENDAD *)
  NUM := 0;
  WHILE NOT EOF(AGENDA) DO
    BEGIN (*1*)
      READ(AGENDA, PERSONA);
      NUM := NUM + 1;
      SEEK(AGENDAD, NUM);
```

```
      WRITE(AGENDAD, PERSONA)
    END; (*1*)
  CLOSE(AGENDA);
  (* Consultas de AGENDAD *)
  WRITE('Número de registro (entre 1 y 100), para terminar 0');
  READLN(NUM);
  WHILE NUM <> 0 DO
    BEGIN (*2*)
      SEEK(AGENDAD, NUM);
      READ(AGENDA, PERSONA);
      WITH PERSONA DO
        WRITELN(NOMBRE, '  ', EDAD, '  ', DIRECCION, '  ', TELEFONO);
      WRITE('Número de registro (entre 1 y 100), para terminar 0');
      READLN(NUM)
    END; (*2*)
  CLOSE(AGENDAD)
END. (*PP*)
```

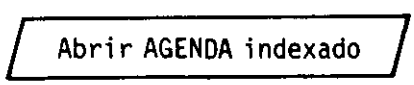
7.6.4. LECTURA-ESCRITURA DE ARCHIVOS INDEXADOS

• **Apertura:** Reserva un archivo indexado en exclusividad para un programa. Si no existía anteriormente lo crea. El archivo queda preparado para lectura, escritura o borrado de sus registros.

Esta organización no está disponible en la mayoría de versiones de los lenguajes **BASIC** y **Pascal**, y en el lenguaje **COBOL** presenta notables diferencias de unas versiones a otras.

Adoptamos la del **BASIC** del sistema operativo **MOS**, y la del **MS-COBOL** del **MS-DOS**, para su estudio. Para utilización de otras versiones se recomienda consultar el manual de referencia correspondiente.

Ordinograma:



Pseudocódigo:

```
abrir AGENDA indexado
```

BASIC: Para la creación de un nuevo archivo indexado, es obligatorio ejecutar, antes de la apertura, la siguiente instrucción de creación:

```
CREATE "K", "AGENDA", NSP, NSE, LRE, KFIELD NL
```

Donde:

"K"	expresa que el archivo es indexado.
"AGENDA"	es su nombre externo.
NSP	es el número de sectores (512 bytes) que se reservan para el área primaria.
NSE	es el número de sectores para el área de excedentes.
LRE	es la longitud del registro *buffer*.
NL	es el número de línea donde está declarado el *buffer* con la sentencia FIELD.

```
OPEN "K", #1, "AGENDA", 72
```

La sentencia FIELD que ha de acompañar a la anterior y que estará en la línea NL, es:

```
FIELD #1, 8 AS DNI$ KEY(1), 20 AS NOMBRE$, 2 AS
    EDAD$, 30 AS DIRECCION$, 12 AS TELEFONO$
```

El campo DNI$ es la *clave primaria*, que no puede estar repetida en dos registros distintos. Esto está expresado mediante la cláusula KEY(1), si hay otras claves secundarias se expresan con KEY(2), KEY(3), etc.

COBOL: Es preciso haber declarado en la cláusula SELECT el soporte, la organización, el modo de acceso y la clave, que ha de ser un campo alfanumérico del registro. Se hace como sigue:

```
SELECT AGENDA ASSIGN TO DISK
              ORGANIZATION IS INDEXED
              ACCESS MODE IS DYNAMIC
              RECORD KEY IS DNI.
```

La apertura del archivo se expresa:

```
OPEN I-O AGENDA.
```

• **Escritura de un registro:** Coloca en una posición del archivo, determinada por el sistema, el contenido del registro *buffer*, al que previamente se le habrán asignado los datos.
La ubicación de cada registro depende del valor asignado al campo clave.

Ordinograma:

Pseudocódigo:

```
escribir AGENDA, PERSONA, DNI
```

BASIC:

```
PUT #1, 1
```

El número 1 indica al sistema la escritura de un registro que no existía previamente.
La asignación previa de datos al registro *buffer*, se hace con instrucciones LSET o RSET.

```
RSET DNI$ = NUMDNI$
```

COBOL:

```
WRITE AGENDA.
```

• **Lectura de un registro:** Coloca el contenido del registro correspondiente a la clave de valor previamente asignado, en el registro *buffer*.

Ordinograma:

Pseudocódigo:

```
leer AGENDA, PERSONA, DNI
```

BASIC:

```
GET #1, KEY(1)
```

Habiendo asignado valor al campo DNI$ con la instruccion RSET o LSET.

COBOL:

```
READ AGENDA.
```

• **Reescritura de un registro:** Coloca en su posición correspondiente, el contenido de un registro que ya existía previamente y cuyos datos han sido modificados.

Ordinograma:

Pseudocódigo:

```
reescribir AGENDA, PERSONA, DNI
```

BASIC:

```
PUT #1, 0
```

El número 0 indica al sistema que el registro con el valor de clave asignado en DNI$, ya existía.

COBOL:

```
REWRITE PERSONA.
```

• **Borrado de un registro:** Elimina un registro existente, determinado por el valor asignado al camplo clave.

Ordinograma:

Pseudocódigo:

```
borrar AGENDA, PERSONA, DNI
```

BASIC:

```
REMOVE #1
```

La clave del registro a borrar ha sido asignada con RSET o LSET.

COBOL:

```
DELETE AGENDA.
```

• **Cierre:** Igual que en los casos anteriores.

En los archivos indexados se puede producir error por las siguientes causas:

— Escritura de un registro que ya existe.
— Lectura de un registro que no existe.
— Reescritura de un registro inexistente.
— Borrado de un registro inexistente.

Estos errores se controlan por medio de:

BASIC: Instrucción

```
ON ERROR GOTO NL
```

Donde NL es el número de línea en donde se continúa la ejecución del programa si se produce algún error.

COBOL: Opción

```
INVALID KEY
```

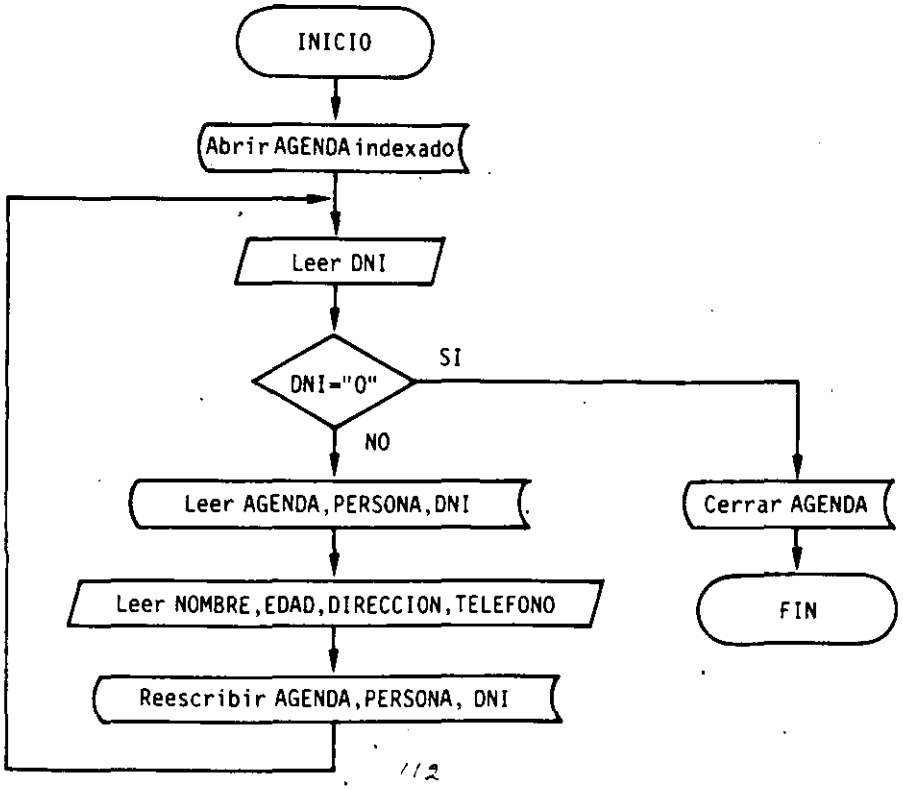
Acompaña a la instrucción de acceso, ejecutándose sus instrucciones en el caso de los errores antes citados. Es obligatorio incluirla en los archivos directos e indexados.

```
WRITE PERSONA INVALID KEY GO TO TRATAR-ERROR.
```

Ejercicio:

Dado el archivo indexado AGENDA que contiene los datos de una serie de personas, cuya clave es el número del DNI. Programa que permite modificar el contenido de algunos de sus registros a partir de datos introducidos por teclado.

Ordinograma:



Búsqueda y clasificación externa

8.0. INTRODUCCION

La característica principal de las aplicaciones de gestión es el manejo de gran cantidad de datos. En la mayoría de los casos, estas aplicaciones manejan varios archivos, siendo las operaciones de búsqueda de las más usuales.

En archivos secuenciales, la lectura de un registro para su consulta, modificación o supresión va precedida necesariamente de la búsqueda del mismo.

En archivos directos, la búsqueda de un registro cuya posición es desconocida, se realiza secuencialmente de la misma forma que en los secuenciales.

Se tiene una situación análoga a la anterior en los archivos indexados, cuando se desea acceder a un registro a partir de un campo que no es la clave, desconociendo el valor de ésta.

Las operaciones de entrada-salida en una aplicación, consumen la mayor parte del tiempo de proceso en comparación con el resto de las operaciones internas, por lo cual interesa que dichas operaciones se reduzcan al mínimo imprescindible. Esta optimización, en el caso de las búsquedas, se consigue si el archivo ha sido previamente clasificado.

Los algoritmos de clasificación de archivos son muy diversos, dependiendo del tamaño y de las características del archivo a clasificar.

En la mayoría de los sistemas operativos actuales, se dispone de programas estándar para realizar la clasificación de un archivo (SORT), no obstante es conveniente conocer el mecanismo de estos algoritmos, por lo que estudiaremos algunos de ellos.

Muchos algoritmos de clasificación externa se basan en la realización de sucesivas particiones y mezclas del archivo a clasificar, por lo que también se presentan los algoritmos básicos para ello.

La mezcla de archivos, además de para clasificarlos, se utiliza para otras aplicaciones, disponiendo los sistemas operativos de programas estándar para realizarla (MERGE).

8.1. BUSQUEDA EN ARCHIVOS SECUENCIALES

Se utiliza un algoritmo de búsqueda cuando se desea obtener el contenido de un registro de un archivo, a partir del valor de uno de sus campos o subcampos, que denominaremos *clave de búsqueda*.

Como se ha mencionado anteriormente, la búsqueda en las organizaciones directa e indexada, cuando sea necesaria, se realizará de forma secuencial, utilizándose los algoritmos que veremos a continuación.

Los algoritmos consisten en un recorrido lineal del archivo, variando la condición de terminación según si el archivo está ordenado o no por la clave de búsqueda.

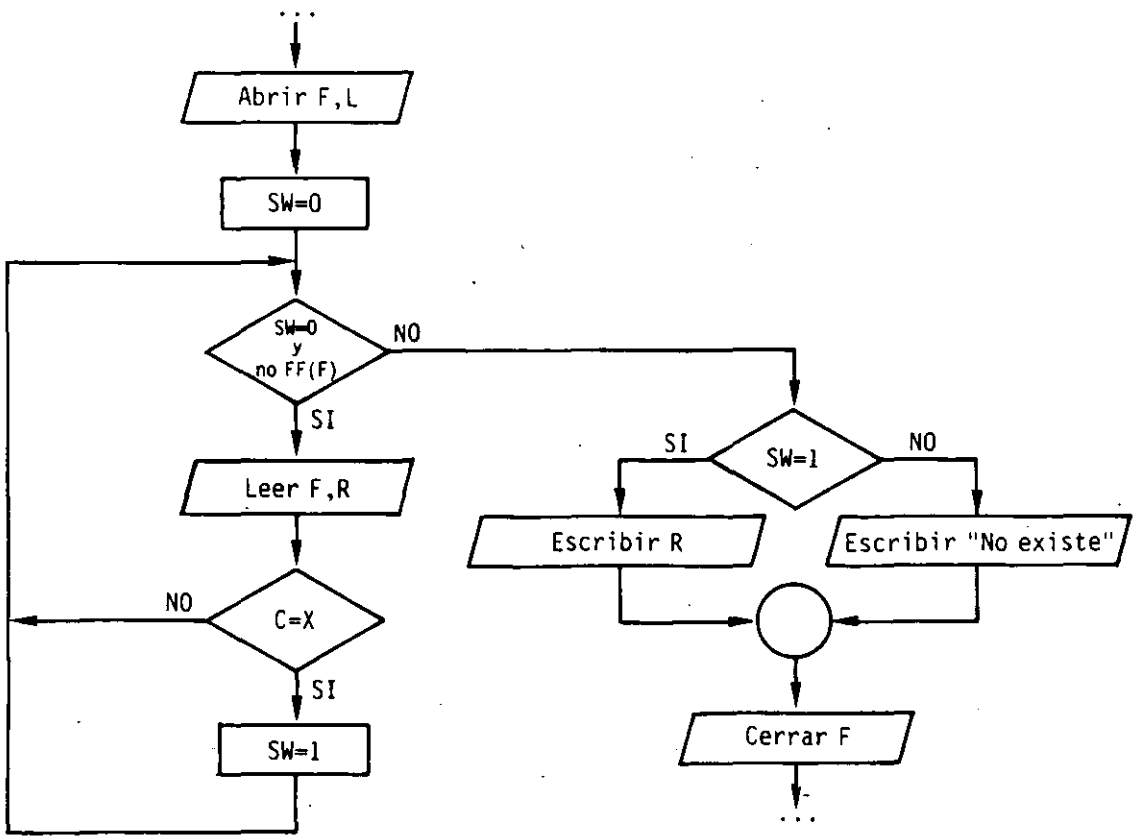
8.1.1. BUSQUEDA EN ARCHIVOS DESORDENADOS

Se recorre el archivo desde el primer registro hasta encontrar aquél cuyo valor de su campo clave coincide con el buscado o hasta que se acabe el archivo, en cuyo caso, se debe indicar la no existencia de dicho registro.

Si existiesen varios registros con el mismo valor del campo clave, igual al buscado, se obtendrá el primero de ellos.

Sea un archivo F, cuyos registros R contienen un campo C, que es la clave de búsqueda, y un valor X a buscar en C.

Ordinograma:

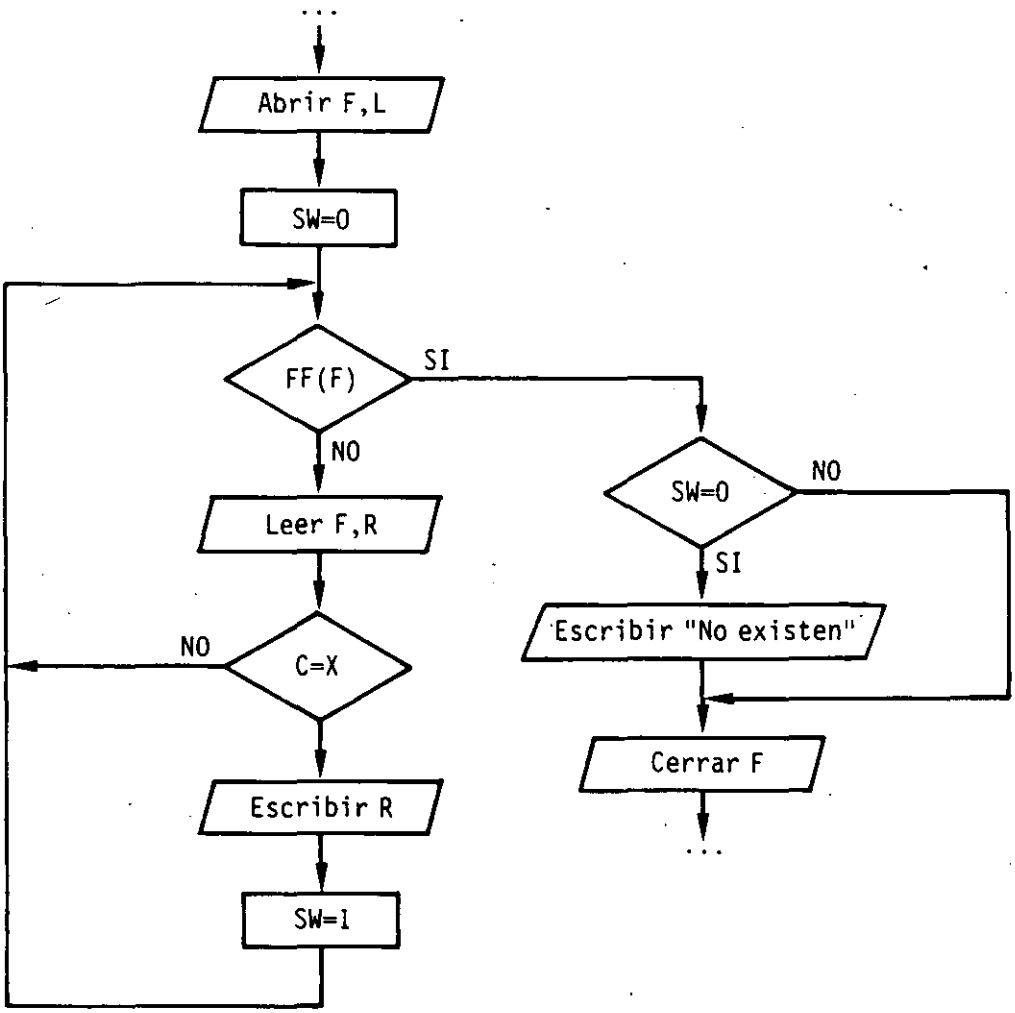


Pseudocódigo:

```
...
abrir F para lectura
SW ← 0
mientras SW = 0 y no FF(F) hacer
  leer F, R
  si C = X entonces SW ← 1 finsi
finmientras
si SW = 1 entonces escribir R
        sino escribir "No existe"
finsi
cerrar F
...
```

Si se desean obtener todos los registros cuyo campo C vale X, se realizará el siguiente algoritmo que recorre F hasta el final:

Ordinograma:



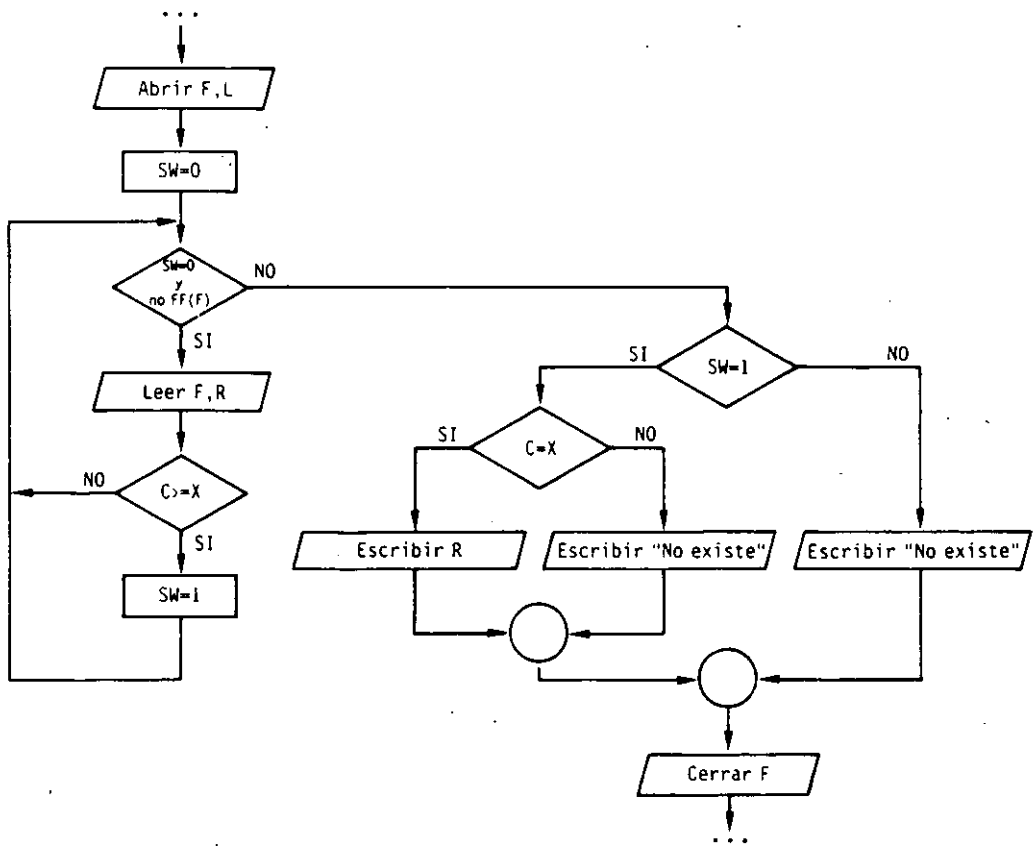
Pseudocódigo:

```
...
abrir F para lectura
SW ← 0
mientras no FF(F) hacer
  leer F, R
  si C = X entonces escribir R; SW ← 1 finsi
finmientras
si SW = 0 entonces escribir "No existen" finsi
cerrar F
...
```

8.1.2. BUSQUEDA EN ARCHIVOS ORDENADOS

Los algoritmos del apartado anterior son válidos para archivos ordenados; no obstante conviene utilizar la ordenación para optimizarlos en cuanto a su tiempo de ejecución, ampliando la condición de terminación de búsqueda al caso de sobrepasar el valor de la clave buscada (supondremos que la ordenación es ascendente).

Ordinograma:

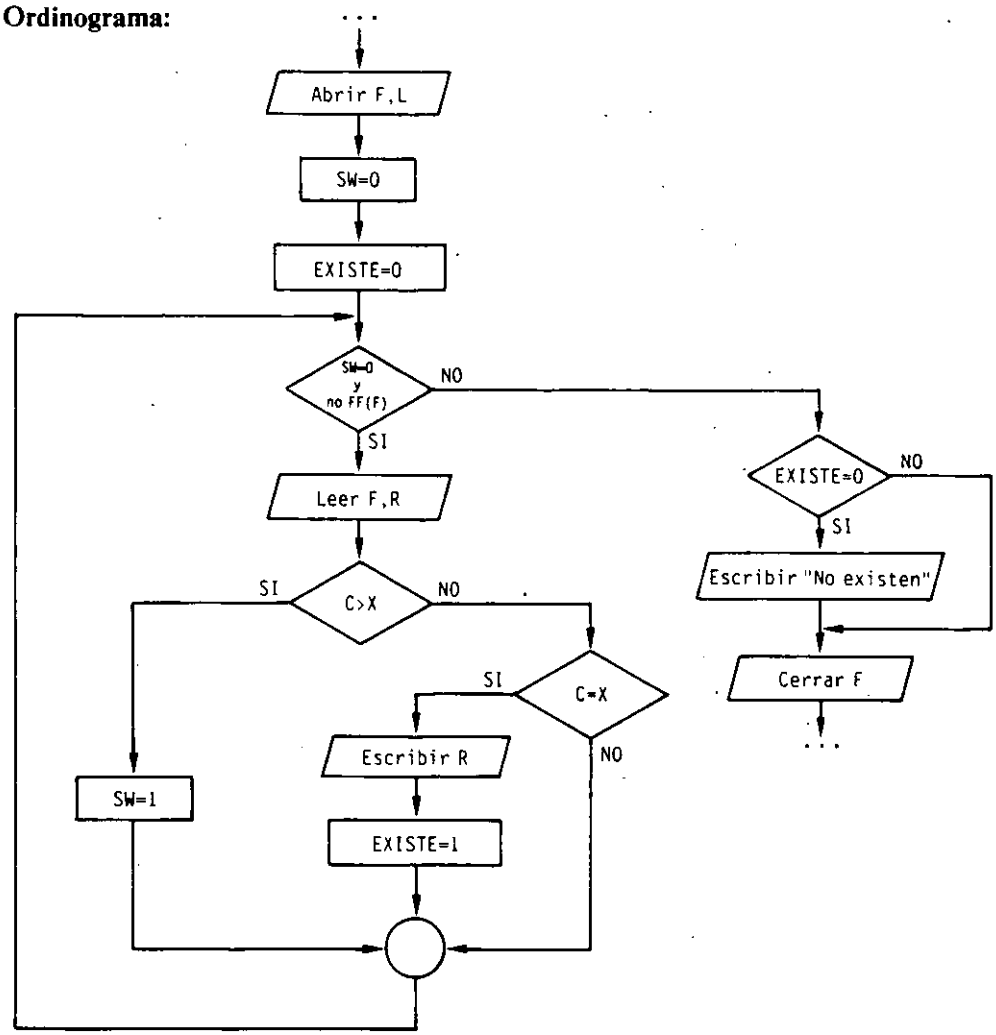


Pseudocódigo:

```
...
abrir F para lectura
SW ← 0
mientras SW = 0 y no FF(F) hacer
  leer F,R
  si C >= X
    entonces SW ← 1
  finsi
finmientras
si SW = 1
  entonces si C = X
              entonces escribir R
              sino escribir "No existe"
            finsi
  sino escribir "No existe"
finsi
cerrar F
...
```

Si se desean obtener todos los registros cuyo campo C vale X, éstos ocuparán posiciones consecutivas en el archivo si existen, terminándose el recorrido al obtenerlos todos.

Ordinograma:



Pseudocódigo:

```
...
abrir F para lectura
SW ← 0
EXISTE ← 0
mientras SW = 0 y no FF(F) hacer
  leer F, R
  si C > X
    entonces SW ← 1
    sino si C = X
           entonces escribir R; EXISTE ← 1
         finsi
  finsi
finmientras
si EXISTE = 0
  entonces escribir "No existen"
finsi
cerrar F
...
```

8.2. PARTICION DE ARCHIVOS

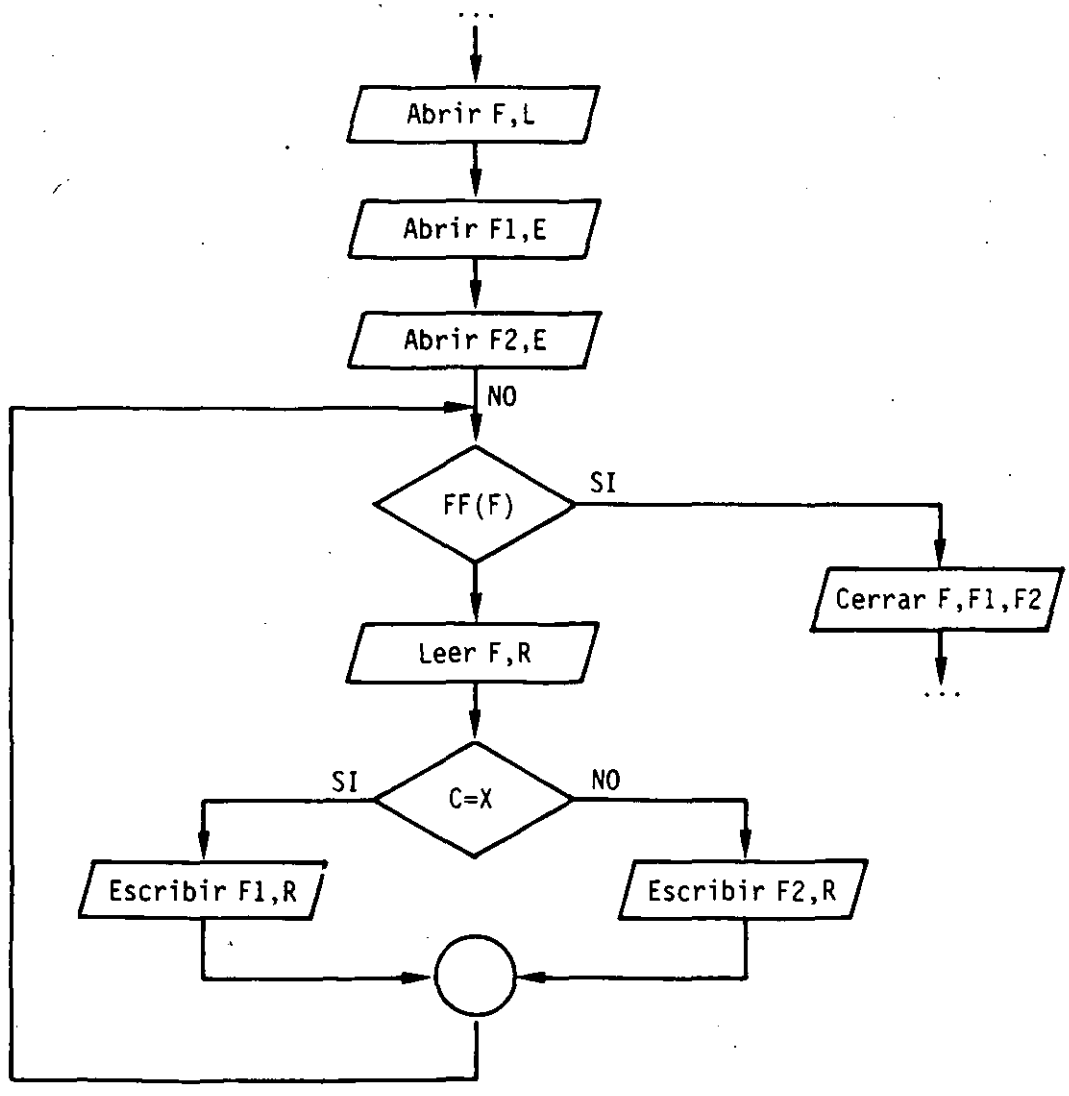
Consiste en repartir los registros de un archivo en otros dos o más, dependiendo de una determinada condición.

8.2.1. PARTICION POR CONTENIDO

La condición la determinan los valores de uno o más campos del registro.

Sea el archivo F, que se desea dividir en dos, F1 y F2, copiando en el primero los registros de F que contienen en el campo C el valor X, y en el segundo los demás.

Ordinograma:



Pseudocódigo:

```
...
abrir F para lectura
abrir F1 para escritura
abrir F2 para escritura
mientras no FF(F) hacer
  leer F, R
  si C = X
    entonces escribir F1, R
    sino     escribir F2, R
  finsi
finmientras
cerrar F, F1, F2
...
```

8.2.2. PARTICION EN SECUENCIAS

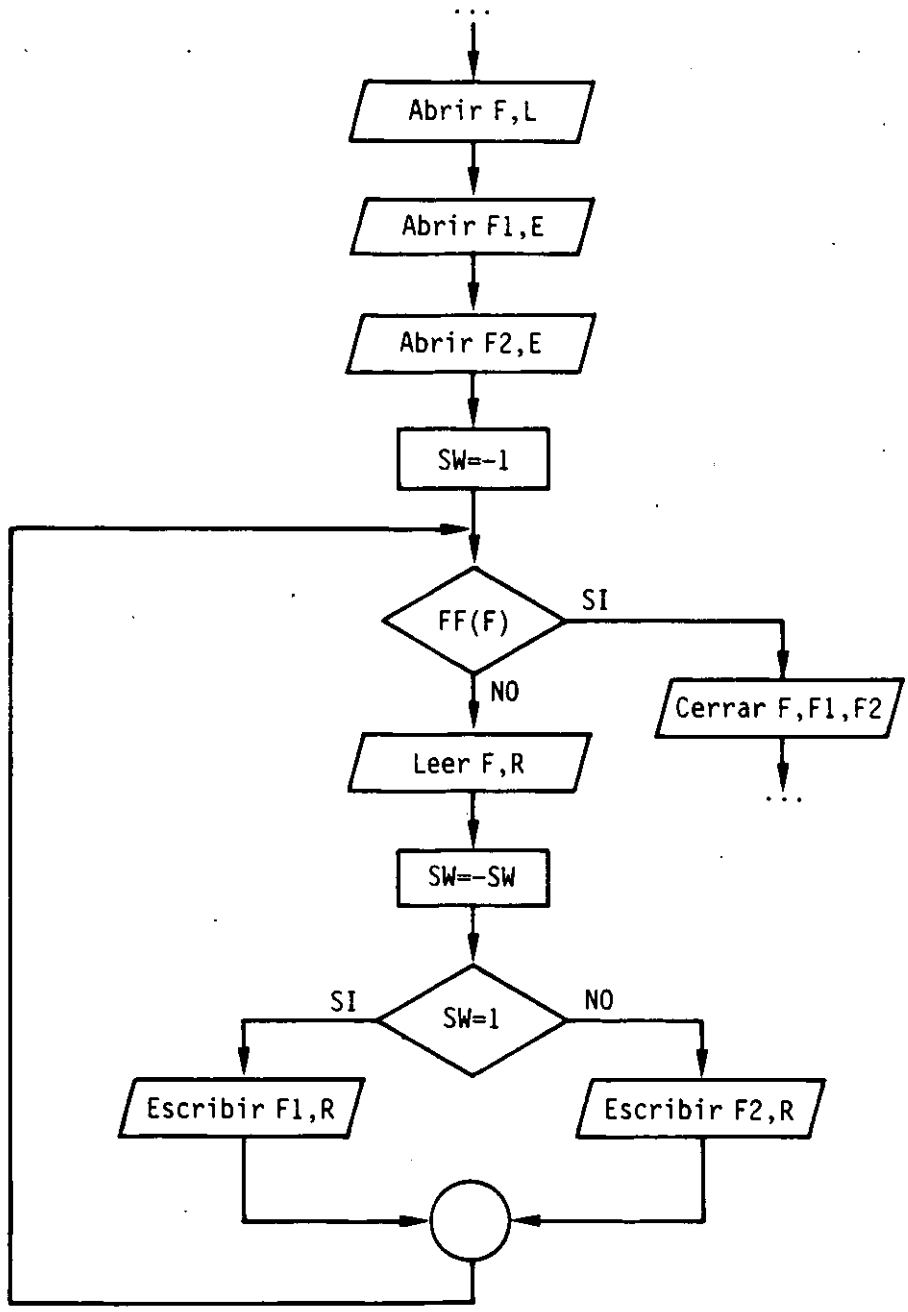
Los registros se distribuyen en secuencias alternativas, de igual o diferente longitud según los casos.

Sea el archivo F, que se desea dividir en dos, F1 y F2, copiando en el primero los registros de F que ocupan posiciones impares, y en el segundo los que ocupan posiciones pares (la longitud de las secuencias es 1).

Pseudocódigo:

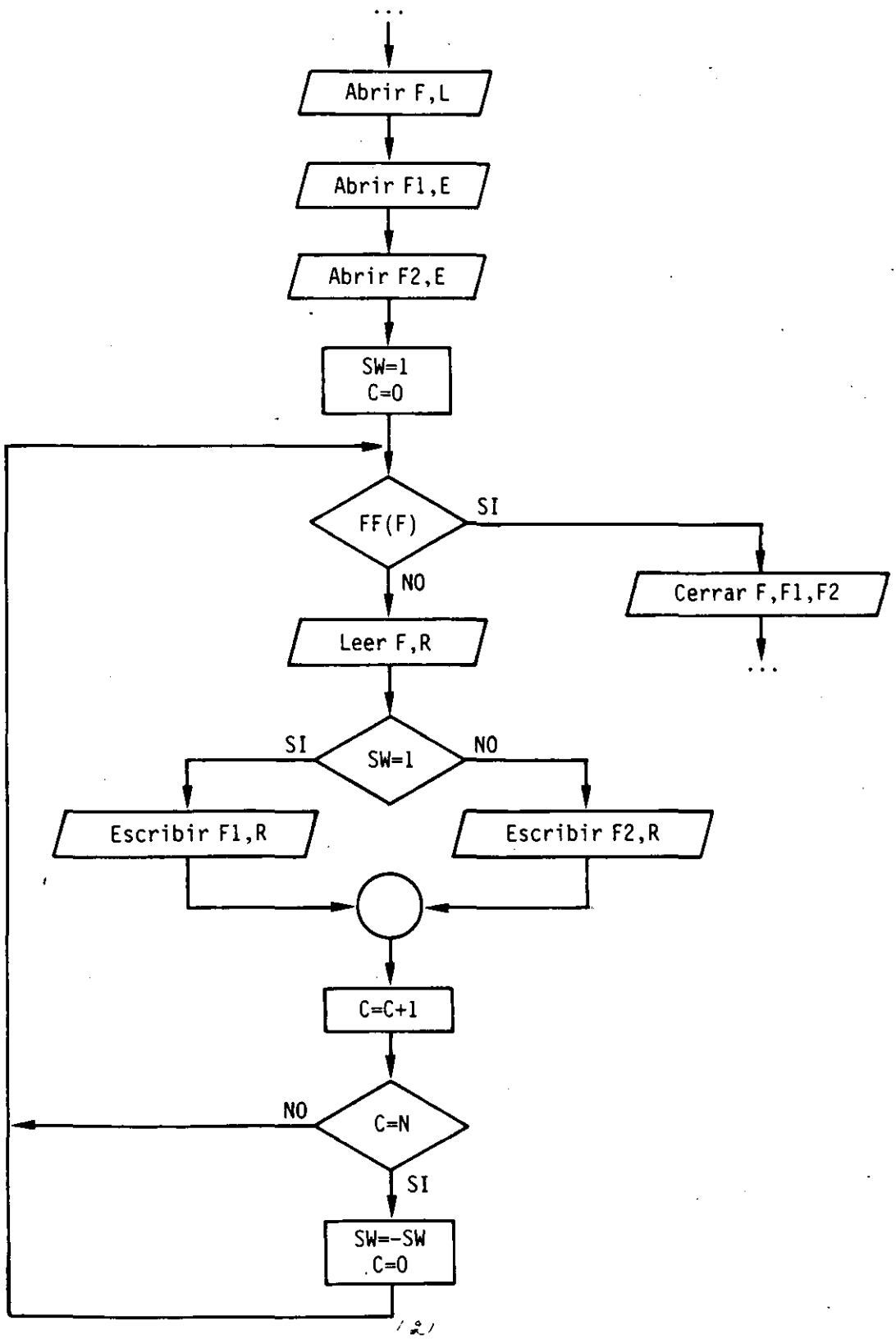
```
...
abrir F para lectura
abrir F1 para escritura
abrir F2 para escritura
SW ← -1
mientras no FF(F) hacer
  leer F, R
  SW ← -SW
  si SW = 1
    entonces escribir F1, R
    sino     escribir F2, R
  finsi
finmientras
cerrar F, F1, F2
...
```

Ordinograma:



Sea el archivo F, que se desea dividir en dos, F1 y F2, copiando alternativamente en uno y otro secuencias de registros de longitud N.

Ordinograma:



Pseudocódigo:

```
...
abrir F para lectura
abrir F1 para escritura
abrir F2 para escritura
SW ← 1
C ← 0
mientras no FF(f) hacer
  leer F, R
  si SW = 1
    entonces escribir F1, R
    sino     escribir F2, R
  finsi
  C ← C + 1
  si C = N
    entonces SW ← -SW; C ← 0
finmientras
cerrar F, F1, F2
...
```

Existen otras particiones con diferentes condiciones y longitudes de secuencia, cuyos algoritmos son similares a los anteriores..

8.3. MEZCLA DE ARCHIVOS

Consiste en reunir en un archivo los registros de dos o más archivos, manteniendo el posible orden que hubiese establecido.

Una mezcla de archivos desordenados consiste en intercalar secuencias de registros de una determinada longitud alternativamente en el archivo destino. Los algoritmos para hacerlo son los inversos de los de partición.

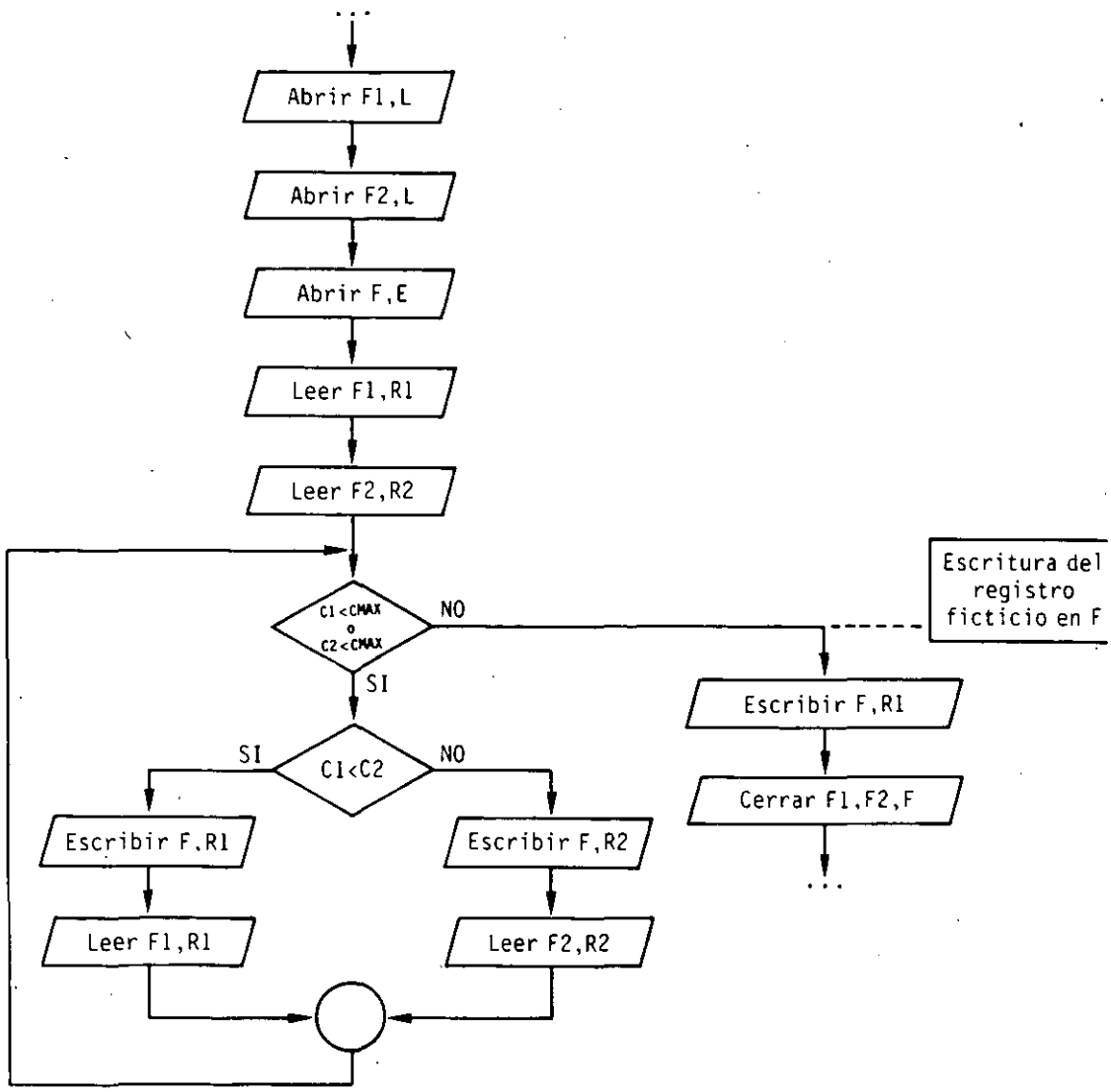
Las mezclas que se estudian a continuación son de aplicación a archivos ordenados ascendentemente por el valor de un campo que denominamos *clave*.

8.3.1. MEZCLA CON REGISTRO CENTINELA

Se dice que un archivo tiene un registro *centinela*, si se le ha añadido un registro al final con el único objetivo de utilizarlo como condición de fin de archivo.

Sean dos archivos F1 y F2, con registros de igual estructura, ordenados por el campo clave C y con un registro centinela de clave máxima CMAX (el valor del campo C en todos los demás registros es inferior al del centinela). Se desea obtener un archivo F ordenado por el mismo campo que contenga los registros de ambos.

Ordinograma:



Pseudocódigo:

```
...
abrir F1 para lectura
abrir F2 para lectura
abrir F para escritura
leer F1, R1
leer F2, R2
mientras C1 < CMAX o C2 < CMAX hacer
  si C1 < C2
    entonces escribir F, R1; leer F1, R1
    sino     escribir F, R2; leer F2, R2
  finsi
finmientras
** escritura del centinela en F
escribir F, R1
cerrar F1, F2, F
...
```

8.3.2. MEZCLA CONTROLADA POR FIN DE ARCHIVO

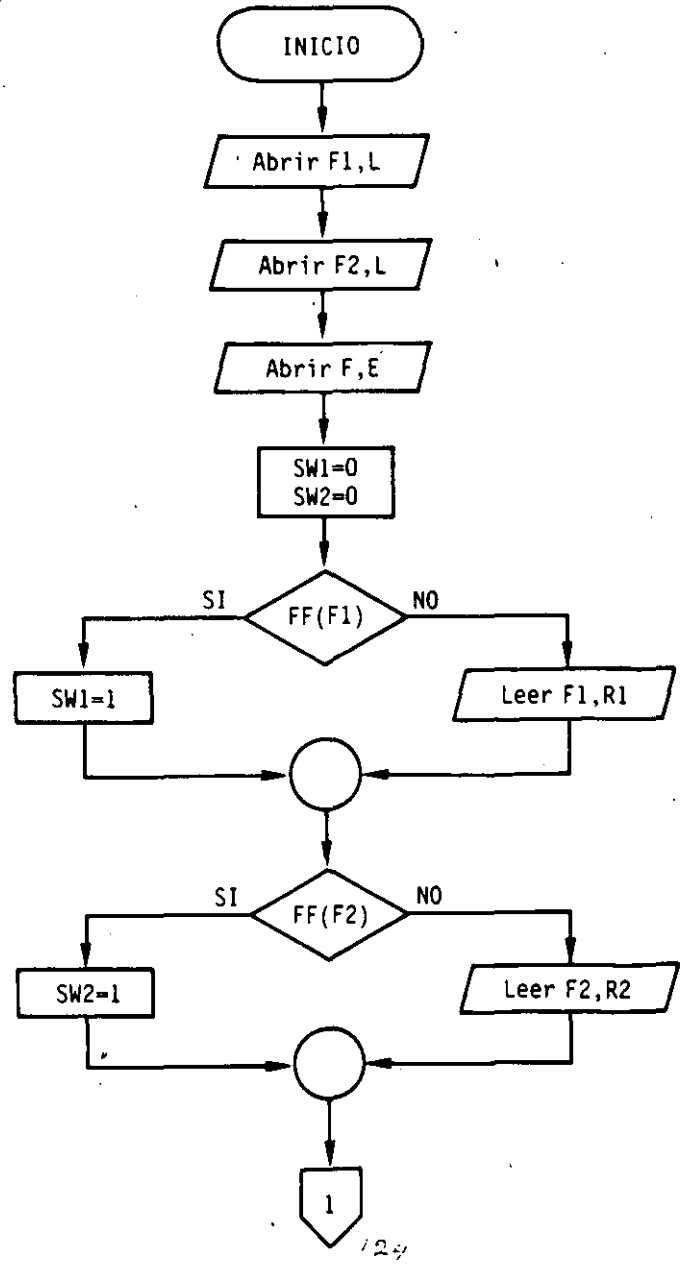
Sean dos archivos F1 y F2, con registros de igual estructura y ordenados por el campo clave C (C1 para el registro R1 de F1 y C2 para el registro R2 de F2). Se desea obtener F ordenado por el mismo campo, que contenga los registros de ambos.

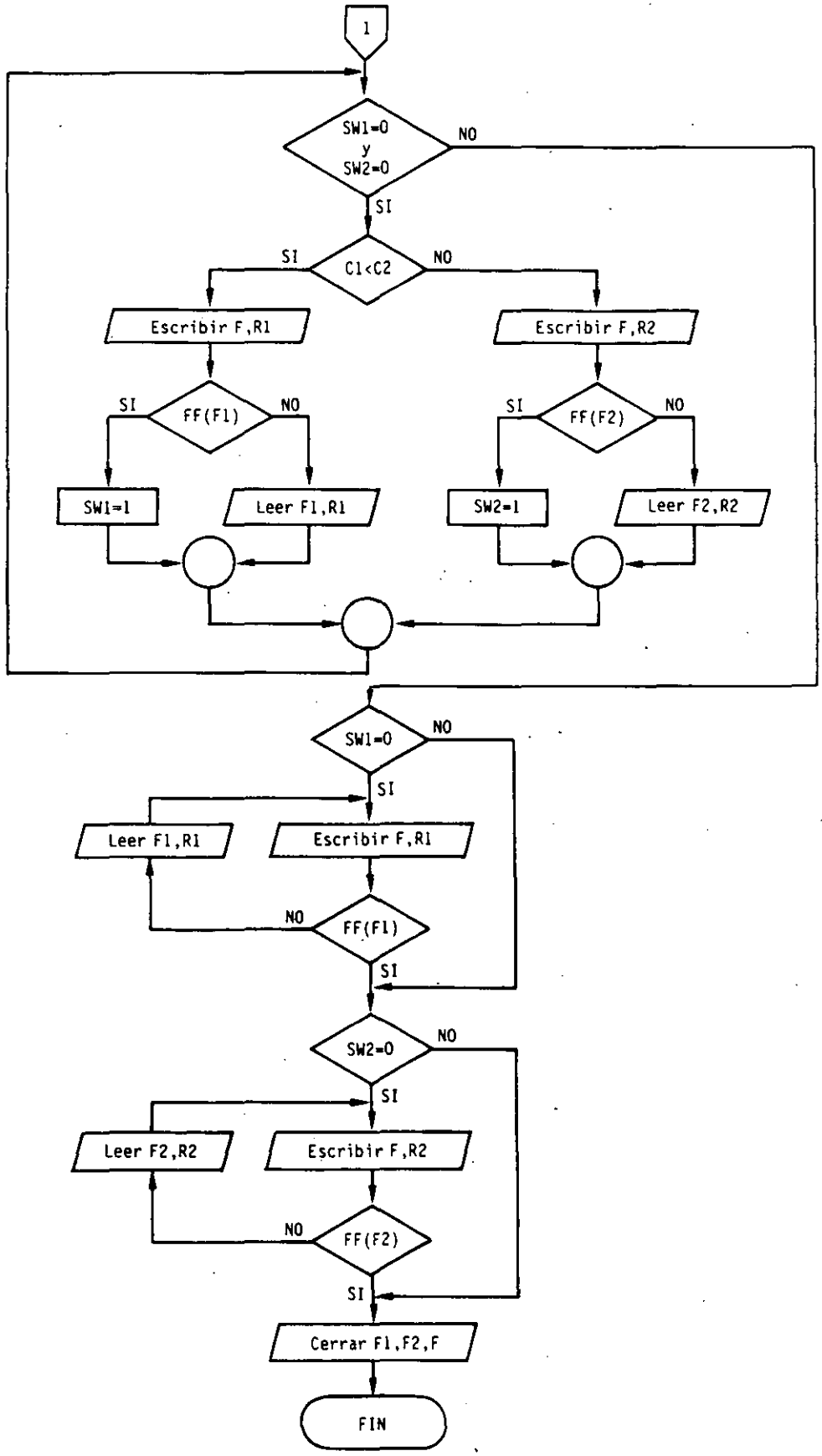
El algoritmo consiste en el recorrido simultáneo de F1 y F2, copiando el registro menor de ambos en F y avanzando en el archivo correspondiente. Se controla la terminación de los archivos por medio de dos *switches* SW1 y SW2.

A la salida del bucle de fusión, uno de los dos archivos no habrá sido copiado en su totalidad, por lo que es necesario añadir dos bucles de copia para completar la fusión con el resto del archivo no copiado. El procesador detecta cuál es el archivo no completado por el valor de su *switch*.

Este algoritmo se puede generalizar para N archivos ordenados, aunque es recomendable, en este caso, realizar sucesivas fusiones de dos en dos, por la complejidad del mismo.

Ordinograma:




1
SW1=0 y SW2=0
NO
SI
C1<C2
SI
NO
Escribir F,R1
Escribir F,R2
FF(F1)
FF(F2)
SI
NO
SW1=1
Leer F1,R1
SW2=1
Leer F2,R2
SW1=0
NO
SI
Leer F1,R1
Escribir F,R1
FF(F1)
NO
SI
SW2=0
NO
SI
Leer F2,R2
Escribir F,R2
FF(F2)
NO
SI
Cerrar F1,F2,F
FIN

Pseudocódigo:

```
...
abrir F1 para lectura
abrir F2 para lectura
abrir F para escritura
SW1 ← 0
SW2 ← 0
si FF(F1)
  entonces SW1 ← 1
  sino leer F1, R1
finsi
si FF(F2)
  entonces SW2 ← 1
  sino leer F2, R2
finsi
mientras SW1 = 0 y SW2 = 0 hacer
  si C1 < C2
    entonces
      escribir F, R1
      si FF(F1)
        entonces SW1 ← 1
        sino leer F1, R1
      finsi
    sino
      escribir F, R2
      si FF(F2)
        entonces SW2 ← 1
        sino leer F2, R2
      finsi
  finsi
finmientras
si SW1 = 0
  entonces
    iterar
      escribir F, R1
      salir si FF(F1)
      leer F1, R1
    finiterar
finsi
si SW2 = 0
  entonces
    iterar
      escribir F, R2
      salir si FF(F2)
      leer F2, R2
    finiterar
finsi
cerrar F1, F2, F
...
```

8.4. CLASIFICACION DE ARCHIVOS

Se dice que un archivo está clasificado ascendente o descendentemente si tiene todos sus registros en secuencia ascendente o descendente respecto al valor de un campo que denominamos *clave de ordenación.*

Un algoritmo de clasificación tiene por objeto redistribuir los registros para que se cumpla la condición de ordenación.

Si el archivo a ordenar, por su tamaño, cabe íntegramente en la memoria central, se carga en una tabla y se realiza una clasificación interna, recopiando el resultado en el archivo original.

Las clasificaciones externas se realizan si el archivo no cabe en la memoria central, teniendo la desventaja de que su tiempo de ejecución es mucho mayor por la gran cantidad de operaciones de entrada y salida que conlleva.

Algunos algoritmos de clasificación son una combinación de ordenación externa e interna, aprovechando al máximo la capacidad de la memoria central.

Se presentan a continuación tres ejemplos de métodos de clasificación externa, de entre los muchos existentes, seleccionados por su sencillez.

8.4.1. CLASIFICACION POR MEZCLA DIRECTA

Se trata de la realización sucesiva de una partición y una mezcla que produce secuencias ordenadas de longitud cada vez mayor.

La primera partición se hace en secuencias de longitud 1, y la fusión correspondiente produce sobre el archivo inicial secuencias ordenadas de longitud 2.

A cada nueva partición y fusión, se duplica la longitud de las secuencias ordenadas.

El proceso termina cuando la longitud de la secuencia ordenada excede la longitud del archivo a ordenar.

Ejemplo:

Sea un archivo F cuyos valores del campo clave son los que figuran a continuación. Se utilizan dos archivos auxiliares F1 y F2 con la misma estructura que F, para realizar las sucesivas particiones y fusiones.

```
F: 15,18,7,75,14,13,43,40,51,93,75,26,64,27,13
```

Partición en secuencias de longitud 1.

```
F1: 15,  7, 14, 43, 51, 75, 64, 13
F2: 18, 75, 13, 40, 93, 26, 27
```

Fusión de secuencias de longitud 1.

```
F: 15,18,  7,75, 13,14, 40,43, 51,93, 26,75, 27,64, 13
```

Partición en secuencias de longitud 2.

```
F1: 15,18, 13,14, 51,93, 27,64
F2:  7,75, 40,43, 26,75, 13
```

Fusión de secuencias de longitud 2.

```
F: 7,15,18,75, 13,14,40,43, 26,51,75,93, 13,27,64
```

Partición en secuencias de longitud 4.

```
F1:  7,15,18,75, 26,51,75,93
F2: 13,14,40,43, 13,27,64
```

Fusión de secuencias de longitud 4.

```
F: 7,13,14,15,18,40,43,75, 13,26,27,51,64,75,93
```

Partición en secuencias de longitud 8.

```
F1:  7,13,14,15,18,40,43,75
F2: 13,26,27,51,64,75,93
```

Fusión de secuencias de longitud 8.

```
F: 7,13,13,14,15,18,26,27,40,43,51,64,75,75,93
```

El proceso se termina al detectarse que la nueva longitud de la secuencia para partición es mayor o igual que el número de registros de F.

8.4.2. CLASIFICACION POR MEZCLA EQUILIBRADA

Es una optimización del método anterior consistente en realizar la partición tomando las secuencias ordenadas de máxima longitud posible y realizando la fusión de secuencias ordenadas alternativamente sobre dos archivos, lo que hace que la siguiente partición quede realizada.

Utiliza tres archivos auxiliares, junto con el original, siendo alternativamente dos de ellos de entrada y los otros dos de salida, para la realización simultánea de fusión y partición.

Durante el proceso de fusión-partición, dos o más secuencias ascendentes, que estén consecutivas, pueden constituir una única secuencia para el paso siguiente.

El proceso termina cuando, en la realización de una fusión-partición, el segundo archivo queda vacío. El archivo totalmente ordenado estará en el primero.

Ejemplo:

Sea F el archivo a ordenar, y F1, F2 y F3 los auxiliares. Con las mismas claves del ejemplo anterior, el proceso es el siguiente.

```
F : 15,18,7,75,14,13,43,40,51,93,75,26,64,27,13
F1:
```

Partición inicial.

```
F2: 15,18, 14, 40,51,93, 26,64, 13
F3:  7,75, 13,43, 75, 27
```

Primera fusión-partición.

```
F :  7,15,18,75, 26,27,64
F1: 13,14,40,43,51,75,93, 13
```

Segunda fusión-partición.

```
F2:  7,13,14,15,18,40,43,51,75,75,93
F3: 13,26,27,64
```

Tercera fusión-partición.

```
F :  7,13,13,14,15,18,26,27,40,43,51,64,75,75,93
F1:
```

8.4.3. CLASIFICACION DE RAIZ

Se utiliza para claves de ordenación numéricas. Consiste en distribuir los registros en diez archivos auxiliares, numerados del 0 al 9, según el dígito de la clave que corresponde al número de pasada que se está realizando. A continuación se concatenan los diez archivos en el archivo original.

Los dígitos de la clave se toman de derecha a izquierda, completándose la ordenación en tantos pasos como dígitos tenga el campo clave.

Ejemplo:

Sea F el archivo de los ejemplos anteriores, y F0, F1, ..., F8, F9 los diez archivos auxiliares. El proceso necesitará dos pasadas compuestas por una partición y una concatenación.

```
F: 15,18,07,75,14,13,43,40,51,93,75,26,64,27,13
```

Primera partición.

```
F0: 40
F1: 51
F2:
F3: 13,43,93,13
F4: 14,64
F5: 15,75,75
F6: 26
F7: 07,27
F8: 18
F9:
```

Primera concatenación.

```
F: 40,51,13,43,93,13,14,64,15,75,75,26,07,27,18
```

Segunda partición.

```
F0: 07
F1: 13,13,14,15,18
F2: 26,27
F3:
F4: 40,43
F5: 51
F6: 64
F7: 75,75
F8:
F9: 93
```

Segunda concatenación.

```
F: 07,13,13,14,15,18,26,27,40,43,51,64,75,75,93
```

8.5. EJERCICIOS PROPUESTOS

E.8.1. Programa que permita consultas a un archivo directo PERSONAS, de 1000 registros, por el contenido de cualquiera de sus campos o subcampos. Los campos son:

NOMBRE alfanumérico
DNI alfanumérico
DOMICILIO, con los subcampos
CALLE alfanumérico
NUMERO numérico entero
POBLACION alfanumérico
PROVINCIA alfanumérico
TELEFONO alfanumérico

La búsqueda se realizará secuencialmente, y en caso de existir repeticiones del valor buscado, se obtendrá el primer registro que lo contenga.

E.8.2. Se dispone de dos archivos secuenciales F1 y F2, ambos con los campos:

CLAVE alfanumérico
RESTO DE CAMPOS

F1 y F2 están ordenados ascendentemente por el campo CLAVE, que es único en cada archivo, existiendo registros comunes a uno y otro. Programa que obtenga el archivo F, intersección de F1 y F2, conteniendo los registros comunes una sola vez.

E.8.3. Programa que dados F1 y F2 del ejercicio anterior, obtenga F3, ordenado por el campo CLAVE, con los registros de ambos que no son comunes (los de F).

E.8.4. Se tiene el archivo secuencial ARTICULOS, cuyos registros contienen los campos: CODIGO DE ARTICULO, PRECIO y EXISTENCIAS, ordenado ascendentemente por el primer campo.

Se tiene el archivo secuencial MOVIMIENTOS, con los campos: TIPO, CODIGO DE ARTICULO, PRECIO y EXISTENCIAS, ordenado ascendentemente por el mismo campo.

El campo TIPO indica la clase de movimiento: "A" para altas, "M" para modificaciones y "B" para bajas (en este último caso sólo se utilizará el valor del campo CODIGO DE ARTICULO).

Programa que genere el archivo NARTICULOS como actualización del archivo ARTICULOS con el de MOVIMIENTOS.

Diseño descendente de programas (top-down)

10.0. INTRODUCCION

Los problemas reales que se plantean a un departamento de informática requieren programas de una cierta complejidad y a veces de gran tamaño.

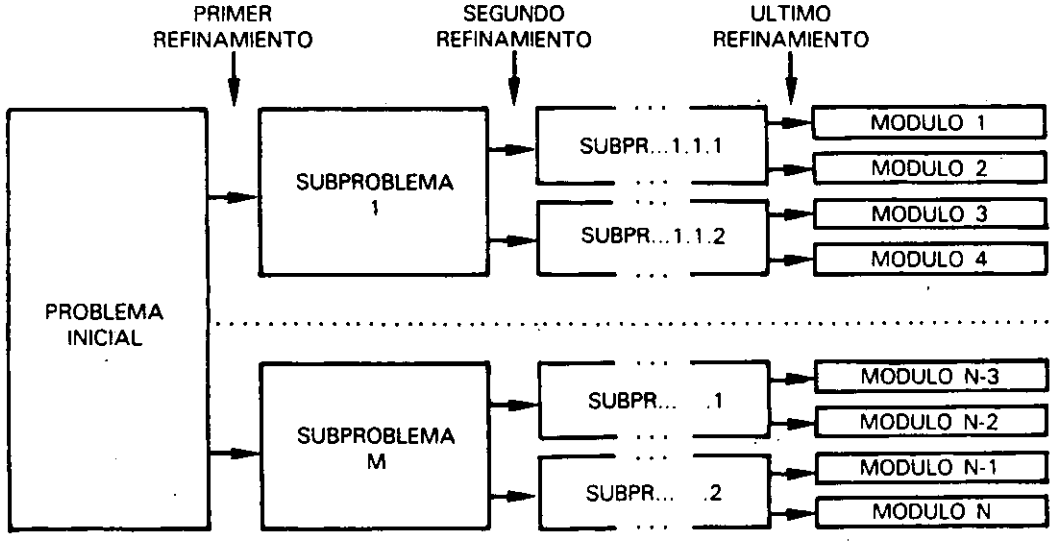
Abordar el diseño de un programa de estas características de una forma directa es una tarea, en la mayoría de los casos, bastante difícil.

Lo más adecuado es descomponer el problema, ya desde su fase de análisis, en partes cuya resolución sea más asequible. La programación de cada una de estas partes se realiza independientemente de las otras, incluso, en ocasiones, por diferentes personas.

De esta forma, se pueden resolver problemas extremadamente complejos.

Por otro lado, la depuración y puesta a punto del programa hace necesario que el listado del mismo sea fácilmente comprensible. En este sentido, conviene subdividir el programa de tal manera que cada parte sea suficientemente reducida y sencilla para su desarrollo y mantenimiento.

El diseño descendente o diseño **top-down** consiste en una serie de descomposiciones sucesivas del problema inicial, que describen el refinamiento progresivo del repertorio de instrucciones que van a formar parte del programa.



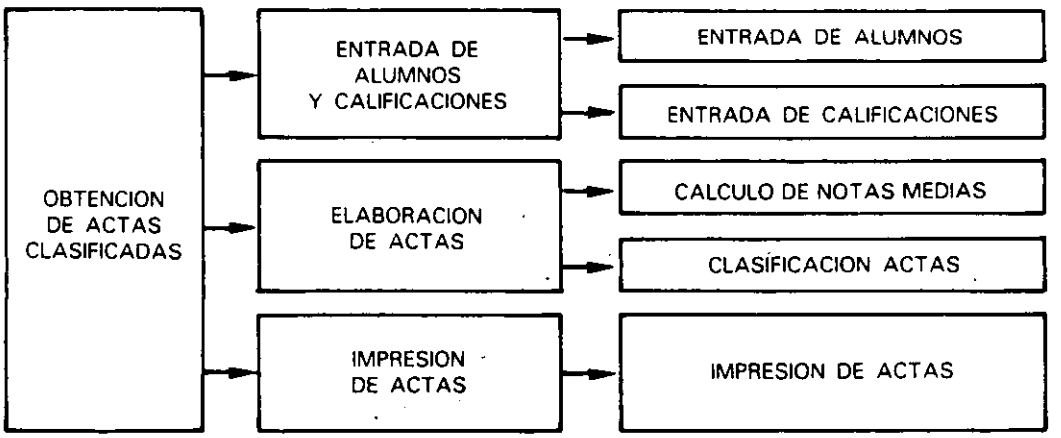
La utilización de esta técnica de diseño, tiene los siguientes objetivos básicos:

— Simplificación del problema y de los subprogramas resultantes de cada descomposición.
— Las diferentes partes del problema pueden ser programadas de modo independiente e incluso por diferentes personas.

— El programa final queda estructurado en forma de *bloques* o *módulos*, lo que hace más sencilla su lectura y mantenimiento.

Ejemplo:

Obtención de las actas de evaluación final, ordenadas alfabéticamente, de los alumnos de un centro docente, a partir de un archivo de alumnos, un archivo de asignaturas y un archivo de calificaciones.



Con la utilización de esta técnica de diseño, surgen los conceptos de:

— **Programa principal y subprogramas.**
— **Subprogramas internos y externos.**
— **Objetos globales y locales.**
— **Parámetros o variables de enlace.**

10.1. PROGRAMA PRINCIPAL Y SUBPROGRAMAS

Un programa diseñado mediante esta técnica quedará constituido por dos partes claramente diferenciadas:

- PROGRAMA PRINCIPAL

Describe la solución completa del problema y consta principalmente de llamadas a subprogramas. Estas llamadas son indicaciones al procesador de que debe continuar la ejecución del programa en el subprograma llamado, regresando al punto de partida una vez lo haya concluido.

El programa principal puede contener, además, instrucciones primitivas y sentencias de control, que son ejecutables de modo inmediato por el procesador.

Un programa principal contendrá pocas líneas, y en él se verán claramente los diferentes pasos del proceso que se ha de seguir para la obtención de los resultados deseados.

- SUBPROGRAMAS

A éstos se les suele denominar *declaración de subprogramas*. Figuran agrupados en distinto lugar al del programa principal.

Su estructura coincide básicamente con la de un programa, con alguna diferencia en el encabezamiento y finalización. En consecuencia, un subprograma puede tener sus propios subprogramas correspondientes a un refinamiento del mismo.

La función de un subprograma es resolver de modo independiente una parte del proble-

ma. Es importante que realice una función concreta en el contexto del problema no obstante, a veces se convierte en subprograma un conjunto de instrucciones, cuando éstas se tendrían que repetir varias veces en diferentes lugares del programa. De esta manera, el conjunto de instrucciones a repetir aparece una sola vez en el listado del programa.

Un subprograma es ejecutado por el procesador exclusivamente al ser llamado por el programa principal o por otro subprograma.

10.2. SUBPROGRAMAS INTERNOS

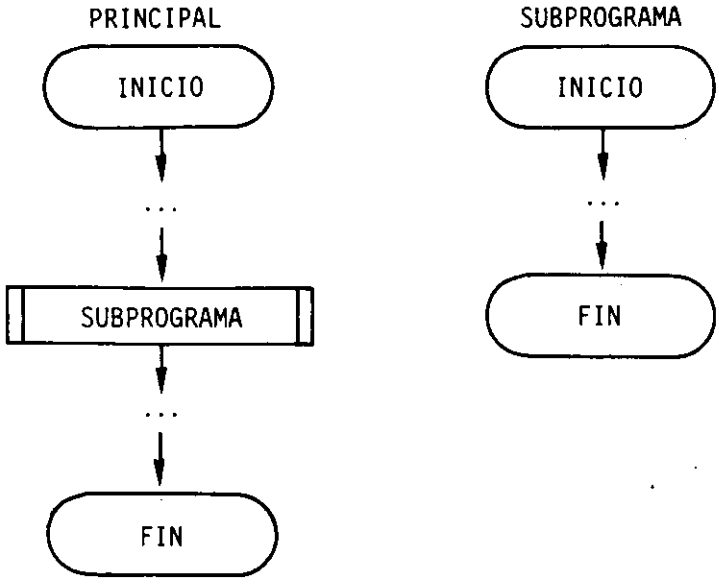
Son subprogramas internos los que figurán junto con el programa principal (en el mismo listado).

Se denominan de diferentes maneras en los distintos lenguajes de programación:

— *Subrutinas* en **BASIC**, que son llamadas mediante la sentencia GOSUB.
— *Párrafos* en **COBOL**, activados mediante sentencias PERFORM.
— *Procedimientos* y *funciones* en **Pascal**, invocados por medio de su *nombre*.

Su representación es la siguiente:

Ordinograma:



Pseudocódigo:

```
Programa PRINCIPAL
Entorno:
  ...
Algoritmo:
  ...
  SUBP
  ...
Finprograma
Subprograma SUBP
Entorno:
  ...
Algoritmo:
  ...
Finsubprograma
```

BASIC:

```
10 REM PRINCIPAL
...
50 GOSUB 500
...
100 END
500 REM SUBP
...
700 RETURN
```

COBOL:

```
IDENTIFICATION·DIVISION.
PROGRAM-ID. PRINCIPAL.
...
PROCEDURE DIVISION.
INICIO.
    ...
    PERFORM SUBP.
    ...
FIN.
    STOP RUN.
SUBP.
    ...
```

Pascal:

```
PROGRAM PRINCIPAL (INPUT, OUTPUT);
  ...
  PROCEDURE SUBP;
  ...
  BEGIN (*SUBP*)
    ...
  END; (*SUBP*)
  ...
BEGIN (*PP*)
  ...
  SUBP;
  ...
END. (*PP*)
```

10.3. SUBPROGRAMAS EXTERNOS

Son aquellos que figuran físicamente separados del programa principal.

Pueden ser compilados separadamente, e incluso pueden haber sido codificados en un lenguaje de programación distinto al del programa principal.

Generalmente, se enlazan con el programa principal en la fase de montaje *(linkage)*, cuando ya son módulos objeto, es decir, traducidos a lenguaje máquina.

Su representación varía de unas versiones a otras de los diferentes lenguajes de programación.

En **ordinograma** y **pseudocódigo** se utiliza la misma representación que para los subprogramas internos.

Las instrucciones de llamada a un subprograma externo son las siguientes:

BASIC:

```
CALLS dico (parámetros)
```

Donde "dico" es la dirección de memoria donde comienza el subprograma llamado.

COBOL:

```
CALL 'SUBPEX' USING parámetros.
```

Siendo "SUBPEX" el nombre del subprograma externo, que ha sido escrito en **COBOL**.

Pascal:

```
EXECUTE(SUBPEX);
```

Siendo "SUBPEX" un archivo declarado en el programa principal y asignado a un archivo externo que contiene un programa en **Pascal**.

10.4. OBJETOS GLOBALES Y LOCALES

Los diferentes objetos que manipula un programa (constantes, variables, tablas, archivos, etcétera) se clasifican según su ámbito, es decir, según la porción de programa y/o subprogramas en que son conocidos, y por tanto pueden ser utilizados.

Son **objetos globales** los declarados en el programa principal, cuyo ámbito se extiende al mismo y a todos sus subprogramas.

Son **objetos locales** a un subprograma, los declarados en dicho subprograma, cuyo ámbito está restringido a él mismo y a los subprogramas declarados en él.

De los lenguajes estudiados, sólo contempla esta clasificación el **Pascal**, para los procedimientos y funciones. Para los lenguajes **COBOL** y **BASIC** todos los objetos son globales.

Ejemplo:

Declaración de variables globales y locales en lenguaje **Pascal:**

```
PROGRAM PRINCIPAL (INPUT, OUTPUT);
  VAR A,B ...
  PROCEDURE SUBP1;
    VAR C,D ...
    PROCEDURE SUBP11;
      VAR E,F ...
      BEGIN (*SUBP11*) ... END; (*SUBP11*)
    PROCEDURE SUBP12;
      VAR G,H ...
      BEGIN (*SUBP12*) ... END; (*SUBP12*)
    BEGIN (*SUBP1*) ... END; (*SUBP1*)
  PROCEDURE SUBP2;
    VAR I,J ...
    BEGIN (*SUBP2*) ... END; (*SUBP2*)
  BEGIN (*PP*)
  ...
  END. (*PP*)
```

135

Variables	Ambito
A, B (globales)	PRINCIPAL, SUBP1, SUBP11, SUBP12, SUBP2
C, D (locales a SUBP1)	SUBP1, SUBP11, SUBP12
E, F (locales a SUBP11)	SUBP11
G, H (locales a SUBP12)	SUBP12
I, J (locales a SUBP2)	SUBP2

10.5. VARIABLES DE ENLACE (PARAMETROS)

Todo programa utiliza unos datos de entrada y produce unos resultados, los primeros provienen de las unidades de entrada, y los segundos son enviados a las unidades de salida, siendo ambas clases de unidades dispositivos externos.

En el caso de un subprograma, los datos de entrada y los resultados provienen y son enviados del y al programa o subprograma llamante, respectivamente.

Para esta labor se utilizan las variables de enlace o parámetros. Es decir, cada vez que se realiza una llamada a un subprograma, los datos de entrada le son pasados por medio de determinadas variables y, análogamente, cuando termina la ejecución del subprograma los resultados regresan mediante otras o mediante las mismas variables.

Los subprogramas también pueden realizar operaciones de entrada y salida con las unidades periféricas cuando sea necesario.

El proceso de emisión y recepción de datos y resultados mediante variables de enlace se denomina **paso de parámetros.**

Los parámetros pueden ser de dos tipos:

— **Parámetros formales.**
— **Parámetros actuales.**

Se denominan parámetros formales a las variables locales de un subprograma utilizadas para la emisión y recepción de los datos.

Se denominan parámetros actuales a las variables y datos enviados, en cada llamada a subprograma, por el programa o subprograma llamante.

Los parámetros formales son siempre fijos para cada subprograma, mientras que los parámetros actuales pueden ser cambiados para cada llamada. En cualquier caso, ha de haber una correspondencia entre los parámetros formales y actuales en su número, colocación y tipo.

El paso de parámetros puede realizarse de dos maneras diferentes:

— Por **valor.**
— Por **referencia.**

El primero se utiliza para suministrar datos de entrada al subprograma, y el segundo para entrada o salida indistintamente.

Un parámetro por valor es un dato o una variable global que contiene un dato de entrada para el subprograma. Esta variable no puede ser modificada por el subprograma, el cual copia su valor en el parámetro formal correspondiente para poder utilizarlo.

Un parámetro por referencia es una variable del programa o subprograma llamante, que puede contener o no un dato para el subprograma llamado, el cual coloca un resultado en esa variable que queda a disposición del llamante una vez concluida la ejecución del subprograma.

Otra importante diferencia entre ambas clases de parámetros, desde el punto de vista físico, consiste en que en el paso por valor, no se proporciona la variable al subprograma, sino solamente su contenido, evitando así su modificación, y en el paso por referencia se proporciona la dirección o referencia de la variable, con lo que el subprograma la utiliza como propia, modificándola si es necesario, para dejar en ella los resultados que ha de devolver.

La utilización de parámetros por referencia supone ahorro de memoria, puesto que la variable local correspondiente no existe físicamente, sino que se asocia a la global en cada llamada. También supone el riesgo de modificar por error una variable global sin desearlo.

En el lenguaje COBOL el paso de parámetros se realiza siempre por referencia con subprogramas externos, los parámetros actuales han de estar declarados en la WORKING-STORAGE SECTION del programa principal y los parámetros formales en la LINKAGE SECTION del subprograma externo llamado. En la sentencia de llamada han de figurar los parámetros actuales, y los formales en la misma línea de la PROCEDURE DIVISION del subprograma llamado, en ambos casos con la opción USING.

10.6. EJERCICIOS RESUELTOS

E.10.1. Programa que gestione de forma interactiva un diccionario INGLES/FRANCES/ESPAÑOL. Para ello se dispone de un archivo secuencial con 1000 registros, conteniendo cada uno tres palabras de igual significado en inglés, francés y español respectivamente.

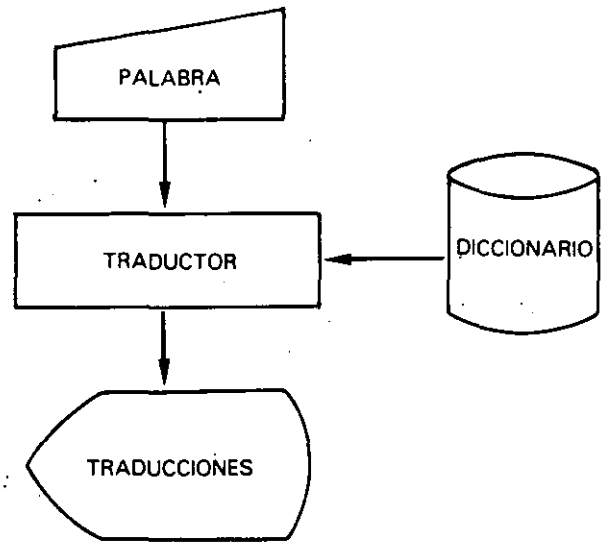
El programa cargará este archivo en una tabla, permitiendo sucesivas consultas a la misma.

El primer dato de entrada indica cuál de los tres diccionarios se desea utilizar en primer lugar. A continuación se introduce una palabra de ese idioma y el programa proporciona sus traducciones, si dicha palabra figura en la tabla.

Este proceso se puede repetir hasta que se desee siendo posible el cambio de un diccionario a otro.

La entrada de datos se realizará mediante elecciones en sucesivos *menus* que se presentarán por pantalla.

Organigrama:



Una vez cargado el archivo en la tabla, la selección de un diccionario se hace sobre la siguiente pantalla:

```
TRADUCTOR DE INGLES/FRANCES/ESPAÑOL

    1. DICCIONARIO DE INGLES
    2. DICCIONARIO DE FRANCES
    3. DICCIONARIO DE ESPAÑOL
    4. TERMINAR
        Escriba opcion: _
```

178

A continuación, se presentará una pantalla según el diccionario elegido, solicitando la palabra a traducir o la vuelta a la pantalla anterior. Por ejemplo, si se eligió la opción número 1, la pantalla es:

```
DICCIONARIO DE INGLES
Escriba la palabra en ingles
que desea traducir o el caracter
"*" para volver al menu principal

Palabra o "*": _
```

Esta pantalla permanecerá hasta que se introduzca el carácter "*". El programa utilizará las líneas inferiores de ésta para imprimir las traducciones solicitadas o un mensaje de que la palabra no está en el diccionario. Por ejemplo, si se introduce la palabra WOMAN, el resultado puede ser una de las dos siguientes pantallas:

```
DICCIONARIO DE INGLES

Escriba la palabra en inglés
que desea traducir o el carácter
"*" para volver al menú principal

Palabra o "*": _
   Inglés:  WOMAN
   Francés: FEMME
   Español: MUJER
```

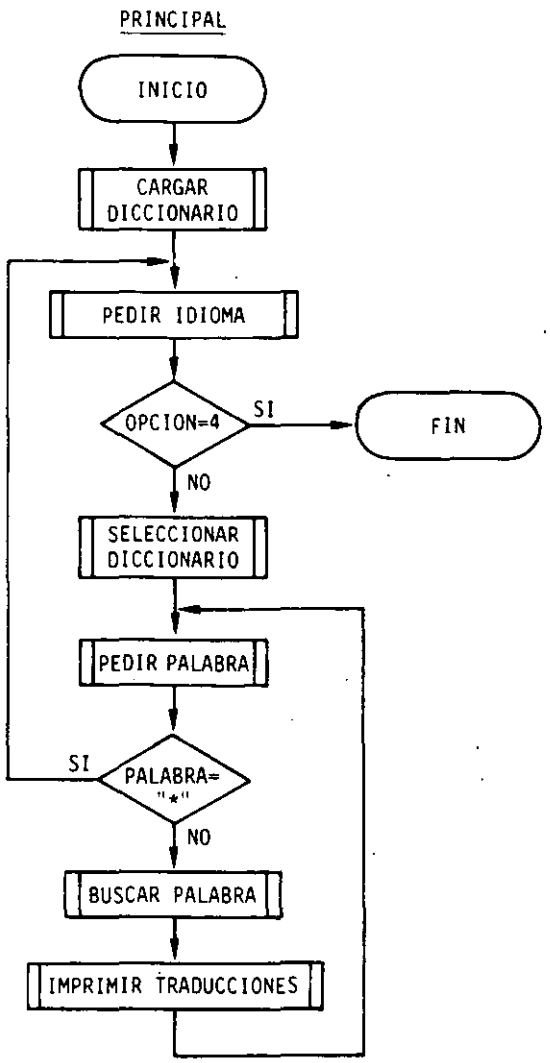
```
DICCIONARIO DE INGLES

Escriba la palabra en inglés
que desea traducir o el caracter
"*" para volver al menú principal

Palabra o "*": _

   La palabra WOMAN no figura
en este diccionario.
```

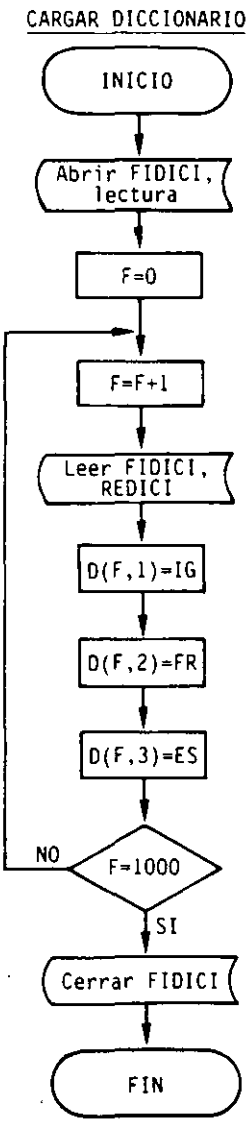
La estructura de datos interna que contiene el diccionario es una tabla D de 1000 filas y 3 columnas, de componentes alfanuméricas, conteniendo en cada fila las 3 palabras de igual significado, la primera en inglés, la segunda en francés y la tercera en español.

D

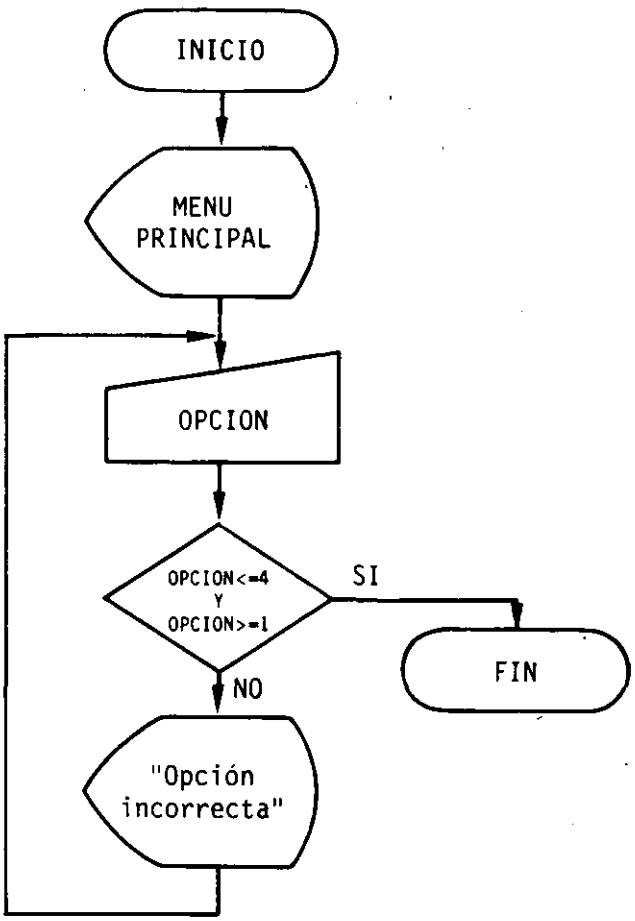
1	OUTPUT	SORTIE	SALIDA
2	INPUT	ENTREE	ENTRADA
3	ARRAY	TABLEAU	TABLA
...	...	...	...
999	COMPUTER	ORDINATEUR	COMPUTADOR
1000	PROGRAM	PROGRAMME	PROGRAMA
	1	2	3

Ordinograma:

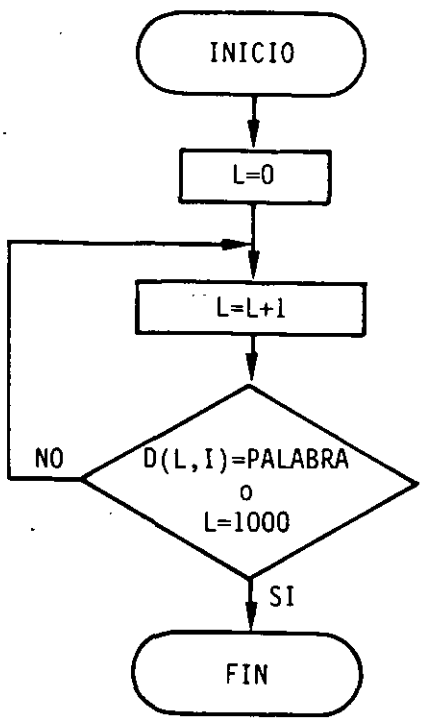




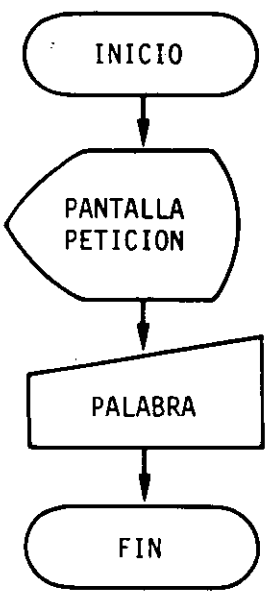
PEDIR IDIOMA



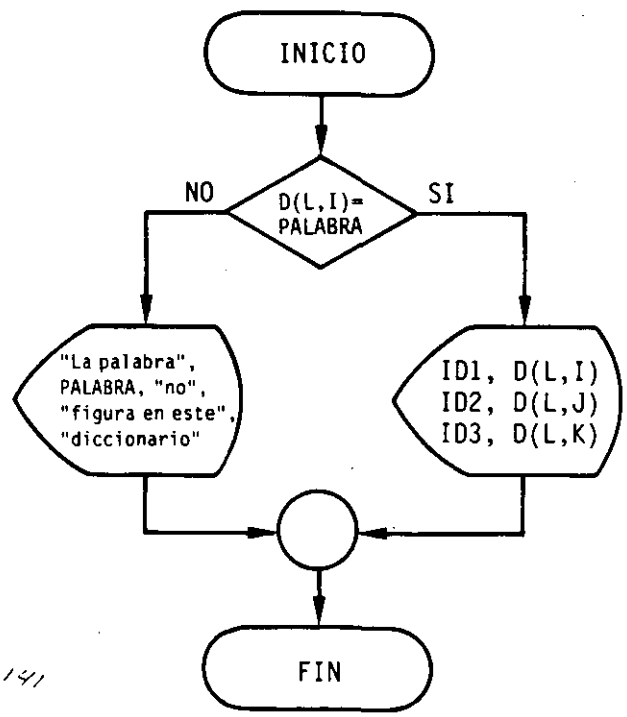
BUSCAR PALABRA



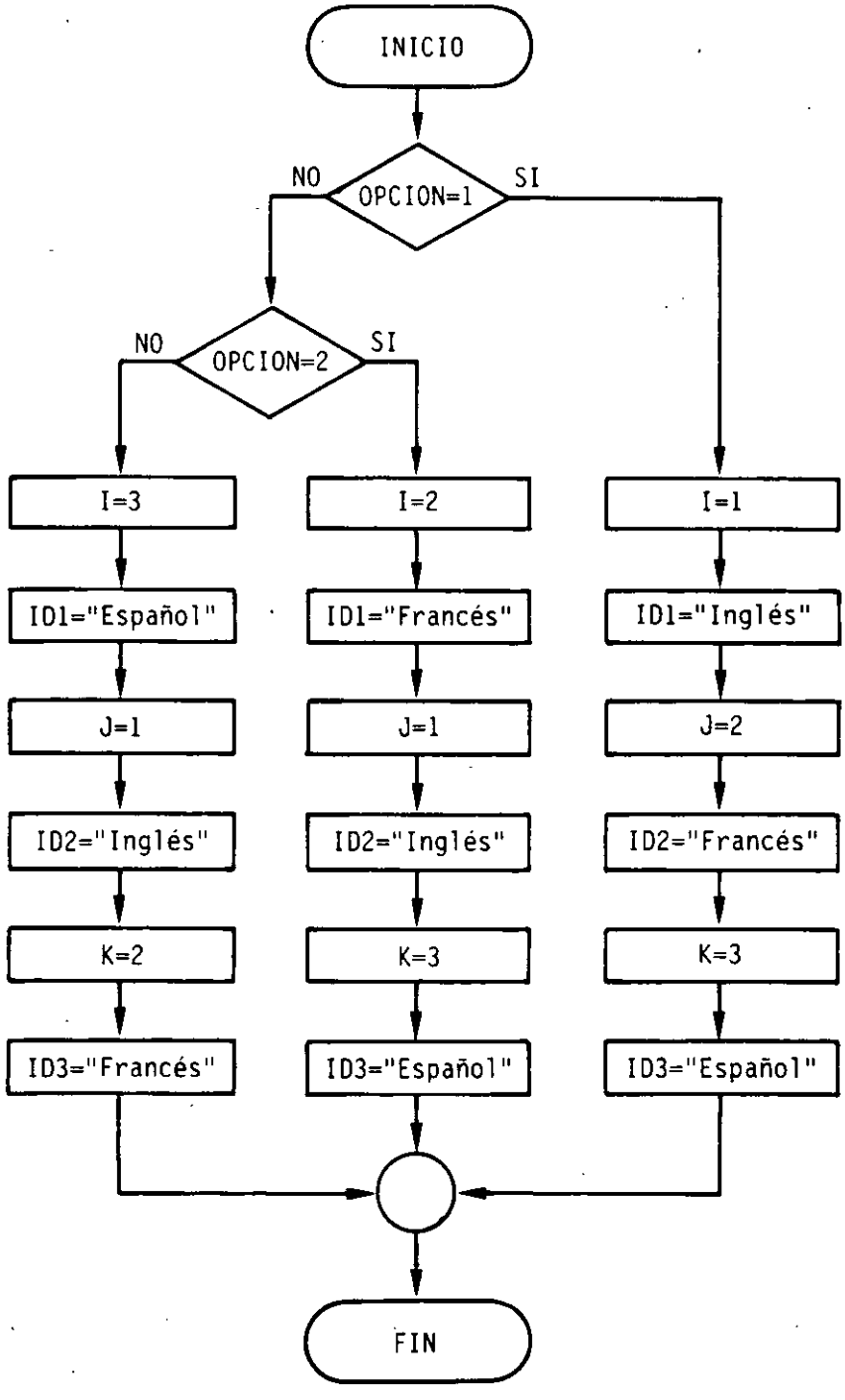
PEDIR PALABRA



IMPRIMIR TRADUCCIONES



SELECCIONAR DICCIONARIO



Pseudocódigo:

```
Programa DICCIONARIO
Entorno:
  D es tabla(1000,3) alfanumérica
  OPCION, I, J, K, L son numéricas enteras
  PALABRA, ID1, ID2, ID3 son alfanuméricas
Algoritmo:
  CARGAR DICCIONARIO
  iterar
    PEDIR IDIOMA
    salir si OPCION = 4
    SELECCIONAR DICCIONARIO
    iterar
      PEDIR PALABRA
      salir si PALABRA = "*"
      BUSCAR PALABRA
      IMPRIMIR TRADUCCIONES
    finiterar
  finiterar
Finprograma
**
Subprograma CARGAR DICCIONARIO
Entorno:
  FIDICI es archivo de REDICI
  REDICI es registro compuesto de
              IG es alfanumérico
              FR es alfanumérico
              ES es alfanumérico
            finregistro
  F es numérica entera
Algoritmo:
  abrir FIDICI para lectura
  para F de 1 a 1000 hacer
    leer FIDICI, REDICI
    D(F,1) ← IG; D(F,2) ← FR; D(F,3) ← ES
  finpara
  cerrar FIDICI
Finsubprograma
**
Subprograma PEDIR IDIOMA
Algoritmo:
  escribir MENU PRINCIPAL
  iterar
    leer OPCION
    salir si 1 <= OPCION <= 4
    escribir "Opción incorrecta"
  finiterar
Finsubprograma
**
```

```
Subprograma SELECCIONAR DICCIONARIO
Algoritmo:
  si OPCION = 1
    entonces I ← 1; ID1 ← "Inglés:  "
             J ← 2; ID2 ← "Francés: "
             K ← 3; ID3 ← "Español: "
    sino si OPCION = 2
           entonces
             I ← 2; ID1 ← "Francés: "
             J ← 1; ID2 ← "Inglés:  "
             K ← 3; ID3 ← "Español: "
           sino
             I ← 3; ID1 ← "Español: "
             J ← 1; ID2 ← "Inglés:  "
             K ← 2; ID3 ← "Francés: "
           finsi
  finsi
Finsubprograma
**
Subprograma PEDIR PALABRA
Algoritmo
  escribir PANTALLA PETICION
  leer PALABRA
Finsubprograma
**
Subprograma BUSCAR PALABRA
Algoritmo:
  L ← 0
  repetir
    L ← L + 1
  hasta D(L,I) = PALABRA o L = 1000
Finsubprograma
**
Subprograma IMPRIMIR TRADUCCIONES
Algoritmo:
  si D(L,I) = PALABRA
    entonces escribir ID1, D(L,I)
             escribir ID2, D(L,J)
             escribir ID3, D(L,K)
    sino escribir "   La palabra ", PALABRA, " no figura"
         escribir "en este diccionario."
  finsi
Finsubprograma
```

Codificación BASIC:

```
10 REM DICCIONARIO
20 DIM D$(1000,3)
30 GOSUB 170 : REM CARGAR DICCIONARIO
40 GOSUB 260 : REM PEDIR IDIOMA
50 IF OPCION = 4 THEN GOTO 150
60 CLS
70 GOSUB 450 : REM SELECCIONAR DICCIONARIO
80 CLS
```

```
90 GOSUB 610 : REM PEDIR PALABRA
100 IF PALABRA$ = "*" THEN GOTO 140
110 GOSUB 730 : REM BUSCAR PALABRA
120 GOSUB 790 : REM IMPRIMIR TRADUCCIONES
130 GOTO 90
140 GOTO 40
150 END
160 REM *****
170 REM CARGAR DICCIONARIO
180 OPEN "FIDICI" FOR INPUT AS #1
190 FOR F = 1 TO 1000
200   INPUT #1, IN$, FR$, ES$
210   D$(F,1) = IN$ : D$(F,2) = FR$ : D$(F,3) = ES$
220 NEXT F
230 CLOSE #1
240 RETURN
250 REM *****
260 REM PEDIR IDIOMA
270 CLS
280 LOCATE 8,22
290 PRINT " TRADUCTOR DE INGLES/FRANCES/ESPAÑOL"
300 PRINT
310 PRINT TAB(22); "     1. DICCIONARIO DE INGLES"
320 PRINT TAB(22); "     2. DICCIONARIO DE FRANCES"
330 PRINT TAB(22); "     3. DICCIONARIO DE ESPAÑOL"
340 PRINT TAB(22); "     4. TERMINAR"
350 PRINT
360 PRINT TAB(22); "         Escriba opción: ";
370 INPUT OPCION
380 IF 1 <= OPCION AND OPCION <= 4 THEN GOTO 430
390 PRINT TAB(22); "         Opción incorrecta"
400 LOCATE 15,47 : PRINT "                    ";
410 LOCATE 15,47
420 GOTO 370
430 RETURN
440 REM *****
450 REM SELECCIONAR DICCIONARIO
460 IF OPCION = 2 THEN GOTO 520
470 IF OPCION = 3 THEN GOTO 560
480 I = 1 : ID1$ = "Inglés:  "
490 J = 2 : ID2$ = "Francés: "
500 K = 3 : ID3$ = "Español: "
510 GOTO 590
520 I = 2 : ID1$ = "Francés: "
530 J = 1 : ID2$ = "Inglés:  "
540 K = 3 : ID3$ = "Español: "
550 GOTO 590
560 I = 3 : ID1$ = "Español: "
570 J = 1 : ID2$ = "Inglés:  "
580 K = 2 : ID3$ = "Francés: "
590 RETURN
600 REM *****
610 REM PEDIR PALABRA
620 LOCATE 8,1
630 PRINT TAB(22); "       DICCIONARIO DE "; ID1$
```

```
640 PRINT
650 PRINT TAB(22); "      Escriba la palabra en ";ID1$
660 PRINT TAB(22); " que desea traducir o el carácter"
670 PRINT TAB(22); " '*' para volver al menú principal"
680 PRINT
690 PRINT TAB(22); "    Palabra o '*': ";
700 INPUT PALABRA$
710 RETURN
720 REM *****
730 REM BUSCAR PALABRA
740 L = 0
750 L = L + 1
760 IF D$(L,I) <> PALABRA$ AND L < 1000 THEN GOTO 750
770 RETURN
780 REM *****
790 REM IMPRIMIR TRADUCCIONES
800 LOCATE 16,1
810 PRINT TAB(28); "                                                    "
820 PRINT TAB(28); "                                                    "
830 PRINT TAB(28); "                                                    "
840 LOCATE 16,1
850 IF D$(L,I) <> PALABRA$ THEN GOTO 900
860 PRINT TAB(31); ID1$; D$(L,I)
870 PRINT TAB(31); ID2$; D$(L,J)
880 PRINT TAB(31); ID3$; D$(L,K)
890 GOTO 930
900 PRINT
910 PRINT TAB(28); "   La palabra "; PALABRA$; " no figura"
920 PRINT TAB(28); "en este diccionario."
930 LOCATE 14,44 : PRINT "                                              "
940 RETURN
```

Codificación COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DICCIONARIO.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FIDICI ASSIGN TO DISK.
DATA DIVISION.
FILE SECTION.
FD  FIDICI LABEL RECORD STANDARD
           VALUE OF FILE-ID 'FIDICI.DAT'.
01  REDICI.
    02 IG PIC X(15).
    02 FR PIC X(15).
    02 ES PIC X(15).
WORKING-STORAGE SECTION.
01  TABLA-D.
    02 FILA-D OCCURS 1000 TIMES.
       03 D PIC X(15) OCCURS 3 TIMES.
77  OPCION PIC 9.
77  I PIC 9.
```

```
 77  J PIC 9.
 77  K PIC 9.
 77  F PIC 9999.
 77  L PIC 9999.
 77  PALABRA PIC X(15).
 77  ID1     PIC X(9).
 77  ID2     PIC X(9).
 77  ID3     PIC X(9).
 SCREEN SECTION.
 01  LIMPIA.
     02 BLANCO BLANK SCREEN.
*    *****
 PROCEDURE DIVISION.
 INICIO.
     PERFORM CARGAR-DIC.
 BLUCLEID.
     PERFORM PEDIR-ID THRU FIN-PEDIRID.
     IF OPCION = 4 GO TO FIN-BLUCLEID.
     DISPLAY LIMPIA.
     PERFORM SELE-DIC.
     DISPLAY LIMPIA.
 BUCLEPAL.
     PERFORM PEDIR-PAL.
     IF PALABRA = '*' GO TO FIN-BUCLEPAL.
     PERFORM BUSCAR-PAL.
     PERFORM IMPRIMIR.
     GO TO BUCLEPAL.
 FIN-BUCLEPAL.
     GO TO BUCLEID.
 FIN-BUCLEID.
     STOP RUN.
*    *****
 CARGAR-DIC.
     OPEN INPUT FIDICI.
     PERFORM CARGA VARYING F FROM 1 BY 1 UNTIL F > 1000.
     CLOSE FIDICI.
 CARGA.
     READ FIDICI.
     MOVE IG TO D(F, 1).
     MOVE FR TO D(F, 2).
     MOVE ES TO D(F, 3).
*    *****
 PEDIR-ID.
     DISPLAY LIMPIA.
     DISPLAY (8, 22) ' TRADUCTOR INGLES/FRANCES/ESPAÑOL'.
     DISPLAY (10, 26) '1. DICCIONARIO DE INGLES'.
     DISPLAY (11, 26) '2. DICCIONARIO DE FRANCES'.
     DISPLAY (12, 26) '3. DICCIONARIO DE ESPAÑOL'.
     DISPLAY (13, 26) '4. TERMINAR'.
     DISPLAY (15, 26) '    Escriba opción: '.

 LECTURA-OP.
     ACCEPT (15, 47) OPCION.
     IF (OPCION NOT < 1) AND (OPCION NOT > 4) GO TO FIN-PEDIRID.
     DISPLAY (16, 26) ' Opción incorrecta'.
```

```
           DISPLAY (15, 47)
           GO TO LECTURA-OP.
       FIN-PEDIRID.
           EXIT.
      *    *****
       SELE-DIC.
           IF OPCION = 1 MOVE 1 TO I, MOVE 'Inglés:  ' TO ID1,
                         MOVE 2 TO J, MOVE 'Francés: ' TO ID2,
                         MOVE 3 TO K, MOVE 'Español: ' TO ID3
             ELSE IF OPCION = 2
                         MOVE 2 TO I, MOVE 'Francés: ' TO ID1,
                         MOVE 1 TO J, MOVE 'Inglés:  ' TO ID2,
                         MOVE 3 TO K, MOVE 'Español: ' TO ID3
                    ELSE
                         MOVE 3 TO I, MOVE 'Español: ' TO ID1,
                         MOVE 1 TO J, MOVE 'Inglés:  ' TO ID2,
                         MOVE 2 TO K, MOVE 'Francés: ' TO ID3.
      *    *****
       PEDIR-PAL.
           DISPLAY (8, 31) 'DICCIONARIO DE ', ID1.
           DISPLAY (10, 30) 'Escriba la palabra en ', ID1.
           DISPLAY (11, 25) 'que desea traducir o el carácter'.
           DISPLAY (12, 25) '"*" para ir al menú principal'.
           DISPLAY (13, 26) '                              '.
           DISPLAY (15, 28) 'Palabra o "*": '.
           ACCEPT  (15, 44) PALABRA.
      *    *****
       BUSCAR-PAL.
           MOVE 1 TO L.
           PERFORM SUMAR UNTIL D(L, I) = PALABRA OR L = 1000.
       SUMAR.
           ADD 1 TO L.
      *    *****
       IMPRIMIR.
           DISPLAY (16, 28) '                              '.
           DISPLAY (17, 28) '                              '.
           DISPLAY (18, 28) '                              '.
           IF D(L, I) = PALABRA DISPLAY (16, 31) ID1, D(L, I),
                                DISPLAY (17, 31) ID2, D(L, J),
                                DISPLAY (18, 31) ID3, D(L, K)
             ELSE DISPLAY (17, 31)  'La palabra ', PALABRA,
                  DISPLAY (18, 28)  'no figura en este diccionario'.
           DISPLAY (15, 44) '
```

Codificación Pascal:

```
PROGRAM DICCIONARIO (INPUT, OUTPUT, FIDICI);
  VAR D : ARRAY[1..1000, 1..3] OF STRING[15];
      OPCION, I, J, K, L : INTEGER;
      PALABRA : STRING[15];
      ID1, ID2, ID3 : STRING[9];
(* ***** *)
  PROCEDURE CARGARDIC;
```

```
    TYPE REG = RECORD
                 IG : STRING[15];
                 FR : STRING[15];
                 ES : STRING[15]
               END;
         FICH = FILE OF REG;
    VAR FIDICI : FICH;
        REDICI : REG;
        F : INTEGER;
    BEGIN (*CARGARDIC*)
      ASSIGN(FIDICI, 'FIDICI.DAT');
      RESET(FIDICI);
      FOR F := 1 TO 1000 DO
        BEGIN (*1*)
          READ(FIDICI, REDICI);
          D[F,1] := REDICI.IG;
          D[F,2] := REDICI.FR;
          D[F,3] := REDICI.ES
        END; (*1*)
      CLOSE(FIDICI)
    END; (*CARGARDIC*)
(* ***** *)
  PROCEDURE PEDIRID;
    BEGIN (*PEDIRID*)
      CLRSCR; (*Borrado de pantalla*)
      GOTOXY(22,8); (*Posicionamiento del cursor*)
      WRITE(' TRADUCTOR INGLES/FRANCES/ESPAÑOL');
      GOTOXY(22,10);
      WRITE('     1. DICCIONARIO DE INGLES');
      GOTOXY(22,11);
      WRITE('     2. DICCIONARIO DE FRANCES');
      GOTOXY(22,12);
      WRITE('     3. DICCIONARIO DE ESPAÑOL');
      GOTOXY(22,13);
      WRITE('     4. TERMINAR');
      GOTOXY(22,15);
      WRITE('         Escriba opción: ');
      READ(OPCION);
      WHILE (OPCION < 1) OR (OPCION > 4) DO
        BEGIN (*2*)
          GOTOXY(22,16);
          WRITE('         Opción incorrecta');
          GOTOXY(47,15);
          READ(OPCION)
        END (*2*)
    END; (*PEDIRID*)
(* ***** *)
  PROCEDURE SELEDIC;
    BEGIN (*SELEDIC*)
      IF OPCION = 1 THEN
             BEGIN (*3*)
               I := 1; ID1 := 'Inglés:  ';
               J := 2; ID2 := 'Francés: ';
               K := 3; ID3 := 'Español: '
             END (*3*)
```

```
        ELSE IF OPCION = 2 THEN
                BEGIN (*4*)
                  I := 2; ID1 := 'Francés: ';
                  J := 1; ID2 := 'Inglés:  ';
                  K := 3; ID3 := 'Español: '
                END (*4*)
             ELSE
                BEGIN (*5*)
                  I := 3; ID1 := 'Español: ';
                  J := 1; ID2 := 'Inglés:  ';
                  K := 2; ID3 := 'Francés: '
                END (*5*)
    END; (*SELEDIC*)
(* ***** *)
  PROCEDURE PEDIRPAL;
    BEGIN (*PEDIRPAL*)
      GOTOXY(22,8);
      WRITE('          DICCIONARIO DE ', ID1);
      GOTOXY(22,10);
      WRITE('        Escriba la palabra en ", ID1);
      GOTOXY(22,11);
      WRITE('   que desea traducir o el carácter');
      GOTOXY(22,12);
      WRITE('   "*" para volver al menú principal');
      GOTOXY(22,14);
      WRITE('      Palabra o "*": ');
      READ(PALABRA);
    END; (*PEDIRPAL*)
(* ***** *)
  PROCEDURE BUSCARPAL;
    BEGIN (*BUSCARPAL*)
      L := 0;
      REPEAT L := L + 1
      UNTIL (D[L,I] = PALABRA) OR (L = 1000)
    END; (*BUSCARPAL*)
(* ***** *)
  PROCEDURE IMPRIMIR;
    BEGIN (*IMPRIMIR*)
      GOTOXY(43,14); CLREOL; (*Borra la línea*)
      GOTOXY(22,16); CLREOL;
      GOTOXY(22,17); CLREOL;
      GOTOXY(22,18); CLREOL;
      IF D[L,I] = PALABRA THEN
         BEGIN (*6*)
           GOTOXY(31,16); WRITE(ID1, D[L,I]);
           GOTOXY(31,17); WRITE(ID2, D[L,J]);
           GOTOXY(31,18); WRITE(ID3, D[L,K])
         END (*6*)
      ELSE
         BEGIN (*7*)
           GOTOXY(31,17); WRITE('La palabra ', PALABRA);
           GOTOXY(31,18); WRITE('no figura en este diccionario.')
         END (*7*)
    END; (*IMPRIMIR*)
(* ***** *)
```

150

```
BEGIN (*PP*)
  CARGARDIC;
  PEDIRID;
  WHILE OPCION <> 4 DO
    BEGIN (*8*)
      CLRSCR; SELEDIC;
      CLRSCR; PEDIRPAL;
      WHILE PALABRA <> '*' DO
        BEGIN (*9*)
          BUSCARPAL;
          IMPRIMIR;
          PEDIRPAL
        END; (*9*)
      PEDIRID
    END (*8*)
END. (*PP*)
```

E.10.2. Programa que realiza el reparto de escaños en unas elecciones por medio de la regla D'HONT.

El programa recibe como datos de entrada:

— Nombre de una circunscripción electoral.
— Censo electoral.
— Número de escaños a distribuir.
— Número de grupos presentados a las elecciones.
— Nombre de cada grupo y su número de votos.
— Número de votos en blanco.
— Número de votos nulos.

El programa proporcionará como resultado la siguiente salida (por pantalla o por impresora):

```
                A C T A   E L E C T O R A L

                Circunscripción:  ..........
Censo electoral:  .......     Número de escaños:  ...
Votos emitidos    .......      ..... % sobre el censo
Abstención        .......      ..... % sobre el censo
Votos válidos     .......      ..... % sobre emitidos
Votos en blanco   .......      ..... % sobre emitidos
Votos nulos       .......      ..... % sobre emitidos

   D I S T R I B U C I O N   D E   E S C A Ñ O S

GRUPO     VOTOS     % CENSO     % EMITIDOS     ESCAÑOS

.....     .....     .....       .....          .....
.....     .....     .....       .....          .....
.....     .....     .....       .....          .....
.....     .....     .....       .....          .....
```

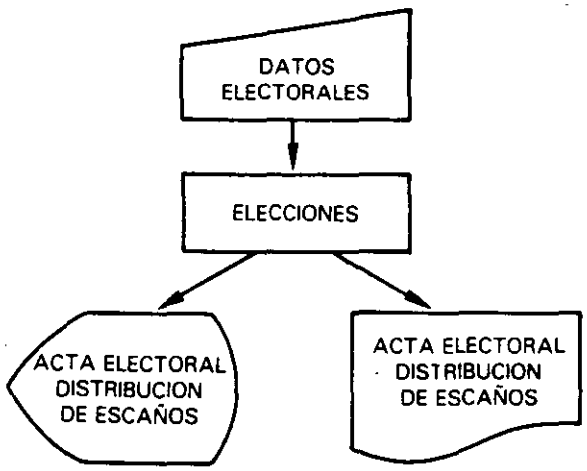
El listado de distribución de escaños se presentará ordenado por número de votos.

El reparto de escaños se realiza utilizando una matriz con una fila por cada grupo, y tantas columnas como número de escaños a repartir.

A cada componente de la matriz se le asigna el número resultante de dividir la cantidad de votos obtenidos por el grupo correspondiente a esa fila, dividido por el número de la columna.

Los escaños se asignan a los grupos en cuyas filas están los máximos valores de la matriz, uno por cada máximo hasta completar el número de escaños.

Organigrama:



La estructura de datos utilizada para el reparto de N escaños entre M grupos, será la matriz:

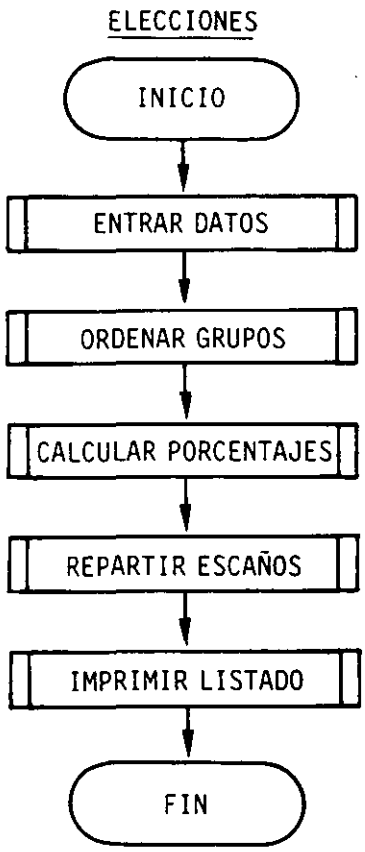
DHONT

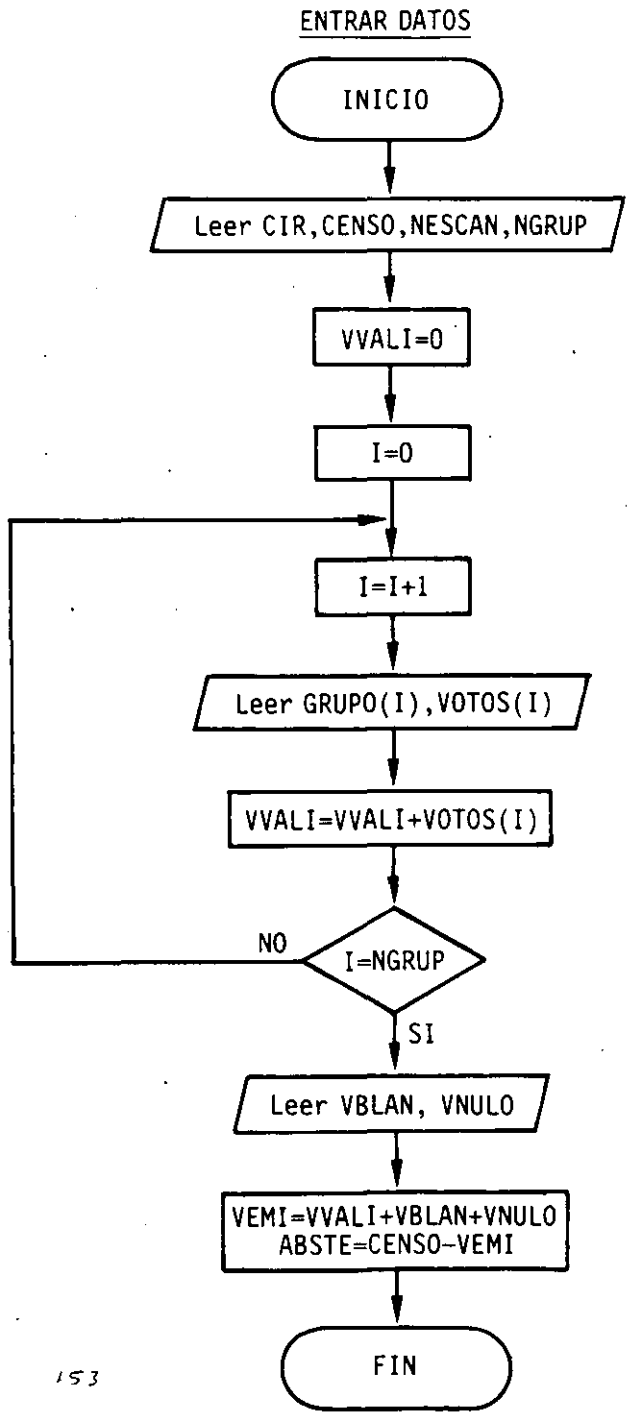
Grupo 1	votos1/1	votos1/2	. . .	votos1/N
Grupo 2	votos2/1	votos2/2	. . .	votos2/N
. . . . .	. . . . .	. . . . .	. . .	. . . . .
Grupo M	votosM/1	votosM/2	. . .	votosM/N
	1	2	. . .	N

Para almacenar los resultados, que habrá que ordenar posteriormente se utilizarán los vectores:

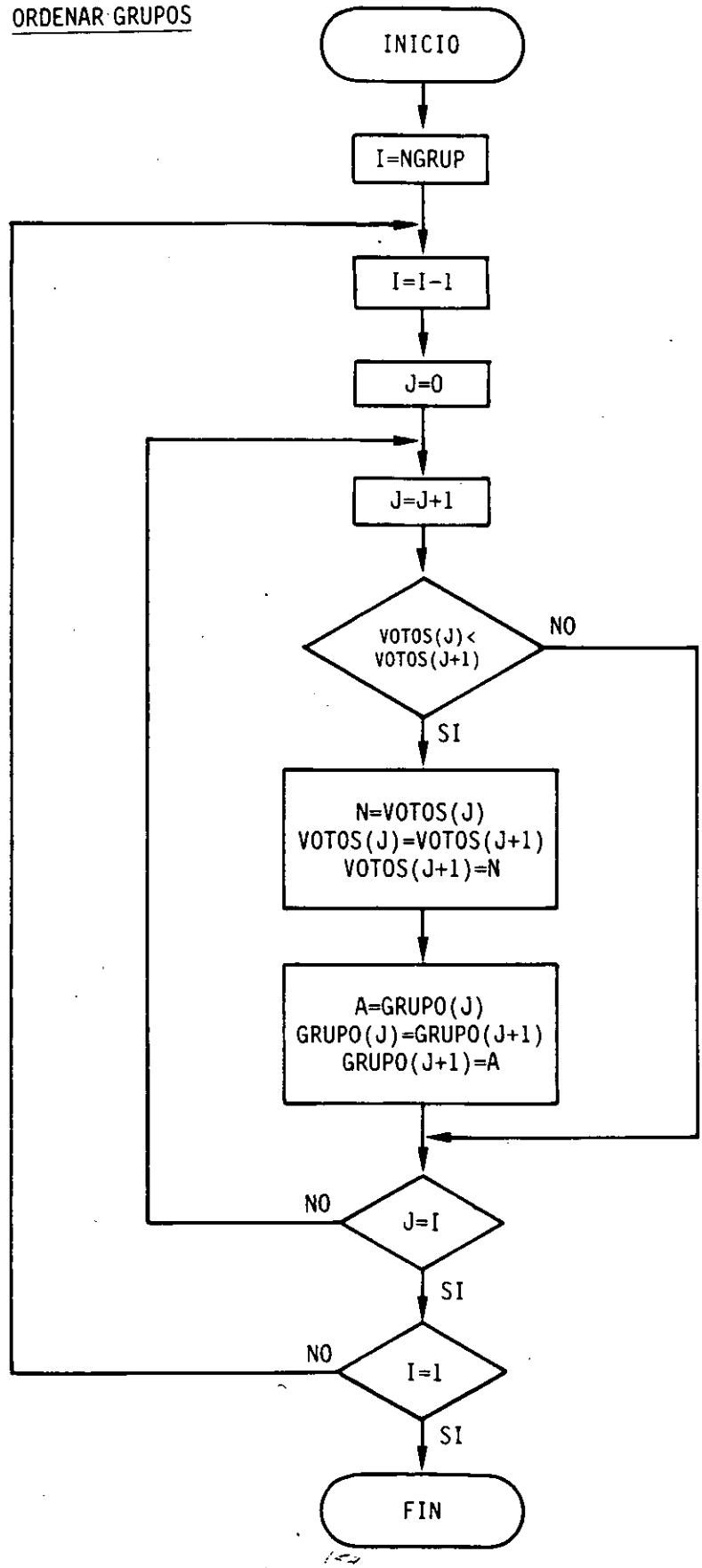
GRUPO	grupo1	grupo2	. . .	grupo M
VOTOS	votos1	votos2	. . .	votosM
PVCEN	pvcen1	pvcen2	. . .	pvcenM
PVEMI	pvemi1	pvemi2	. . .	pvemiM
ESCAN	escan1	escan2	. . .	escanM
	1	2	. . .	M

Ordinograma:

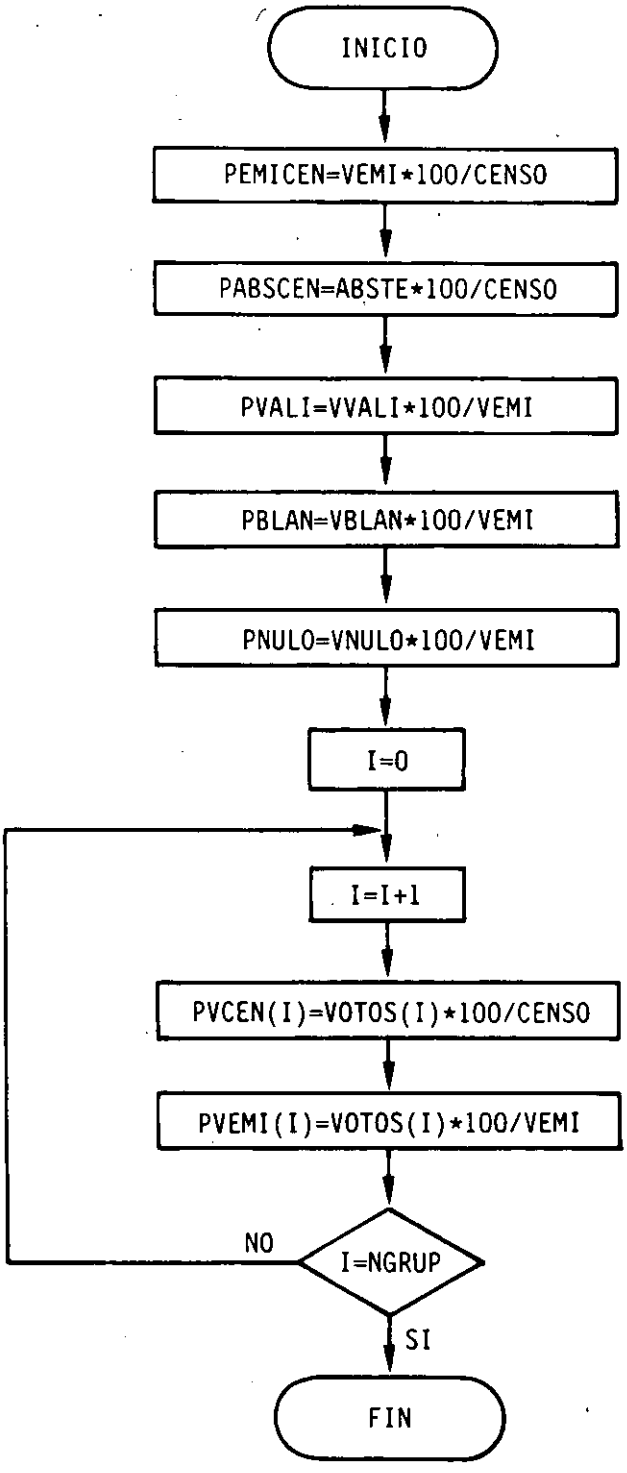




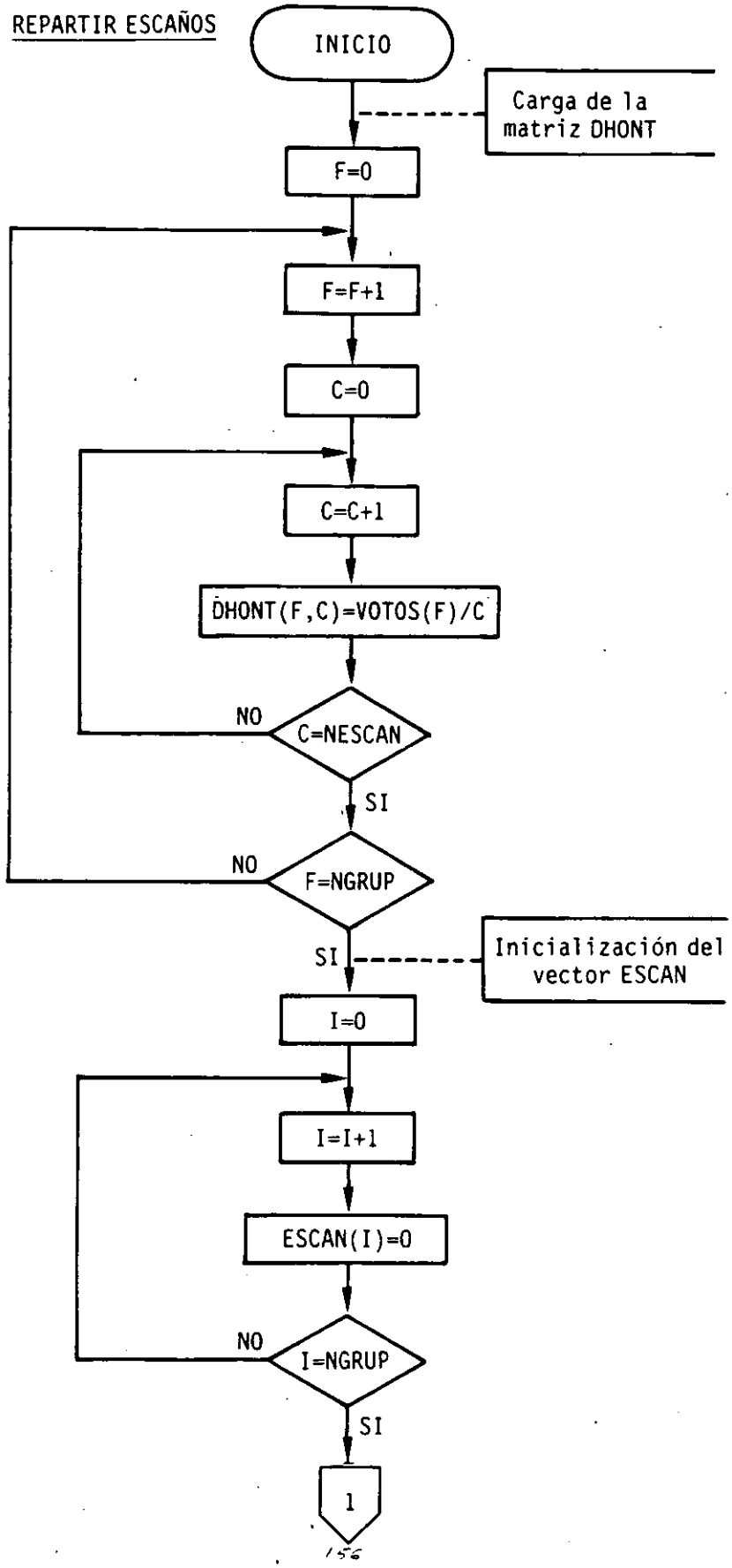
ORDENAR GRUPOS

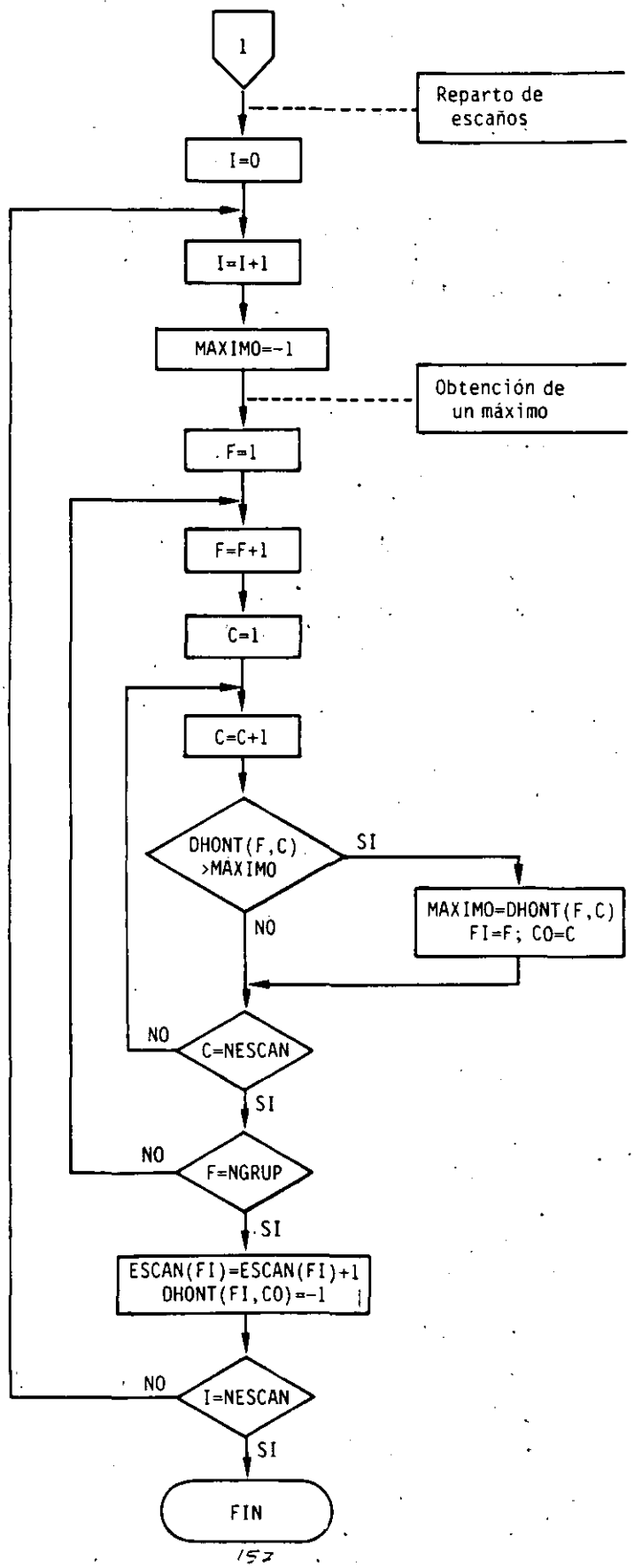


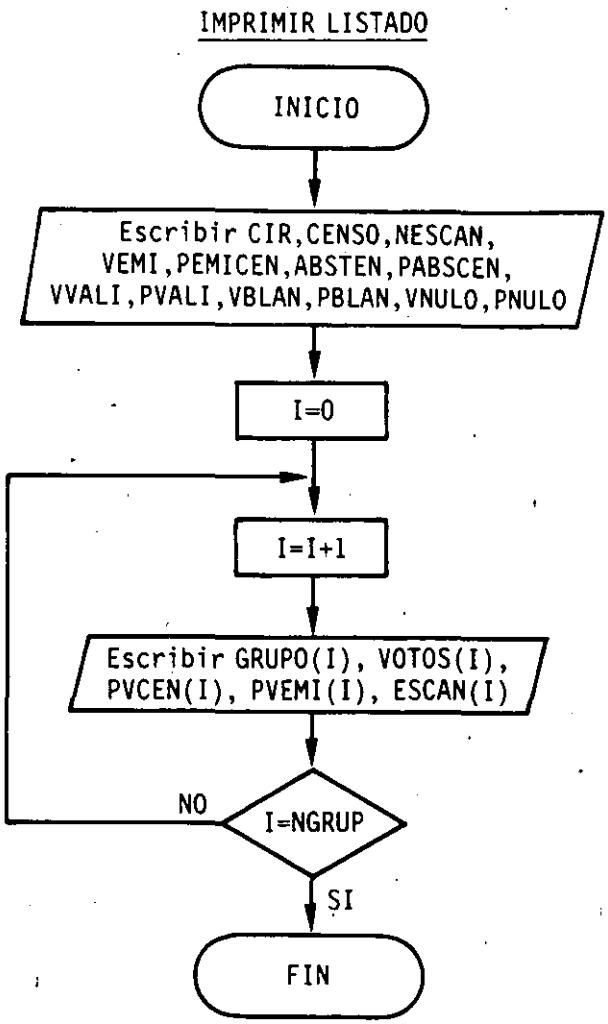
CALCULAR PORCENTAJES



155


REPARTIR ESCAÑOS
INICIO
Carga de la matriz DHONT
F=0
F=F+1
C=0
C=C+1
DHONT(F,C)=VOTOS(F)/C
NO
C=NESCAN
SI
NO
F=NGRUP
SI
Inicialización del vector ESCAN
I=0
I=I+1
ESCAN(I)=0
NO
I=NGRUP
SI
1


1
Reparto de escaños
I=0
I=I+1
MAXIMO=-1
Obtención de un máximo
F=1
F=F+1
C=1
C=C+1
DHONT(F,C) >MAXIMO
SI
NO
MAXIMO=DHONT(F,C)
FI=F; CO=C
NO
C=NESCAN
SI
NO
F=NGRUP
SI
ESCAN(FI)=ESCAN(FI)+1
DHONT(FI,CO)=-1
NO
I=NESCAN
SI
FIN



Pseudocódigo:

```
Programa ELECCIONES
Entorno:
  DHONT es tabla(15,40) numérica real
  ** Máximo de 15 grupos y 40 escaños
  GRUPO es tabla(15) alfanumérica
  ** Nombres de los grupos participantes
  VOTOS es tabla(15) numérica entera
  ** Número de votos de cada grupo
  PVCEN es tabla(15) numérica real
  ** Porcentaje de votos de cada grupo sobre el censo
  PVEMI es tabla(15) numérica real
  ** Porcentaje de votos de cada grupo sobre emitidos
  ESCAN es tabla(15) numérica entera
  ** Número de escaños obtenidos por cada grupo
  CIR es alfanumérica ** Nombre de la circunscripción
  CENSO, NESCAN, NGRUP, VEMI, ABSTE, VVALI, VBLAN,
  VNULO, I, J son numéricas enteras
  PEMICEN, PABSCEN, PVALI, PBLAN, PNULO son numéricas reales
```

```
Algoritmo:
  ENTRAR DATOS
  ORDENAR GRUPOS
  CALCULAR PORCENTAJES
  REPARTIR ESCAÑOS
  IMPRIMIR LISTADO
Finprograma
Subprograma ENTRAR DATOS
Algoritmo:
  leer CIR, CENSO, NESCAN, NGRUP
  VVALI ← 0
  para I de 1 a NGRUP hacer
    leer GRUPO(I), VOTOS(I)
    VVALI ← VVALI + VOTOS(I)
  finpara
  leer VBLAN, VNULO
  VEMI ← VVALI + VBLAN + VNULO
  ABSTE ← CENSO - VEMI
Finsubprograma
Subprograma ORDENAR GRUPOS
** Ordena simultáneamente los vectores GRUPO y VOTOS
** en orden descendente según los valores del segundo
Entorno:
  A es alfanumérica
  N es numérica entera
Algoritmo:
  para I de NGRUP - 1 a 1 con incremento -1 hacer
    para J de 1 a I hacer
       si VOTOS(J) < VOTOS(J + 1) entonces
         N ← VOTOS(J); VOTOS(J) ← VOTOS(J + 1)
         VOTOS(J + 1) ← N; A ← GRUPO(J)
         GRUPO(J) ← GRUPO(J + 1); GRUPO(J + 1) ← A
       finsi
    finpara
  finpara
Finsubprograma
Subprograma CALCULAR PORCENTAJES
Algoritmo:
  PEMICEN ← VEMI * 100 / CENSO
  PABSCEN ← ABSTE * 100 / CENSO
  PVALI ← VVALI * 100 / VEMI
  PBLAN ← VBLAN * 100 / VEMI
  PNULO ← VNULO * 100 / VEMI
  para I de 1 a NGRUP hacer
    PVCEN(I) ← VOTOS(I) * 100 / CENSO
    PVEMI(I) ← VOTOS(I) * 100 / VEMI
  finpara
Finsubprograma
Subprograma REPARTIR ESCAÑOS
Entorno:
  MAXIMO es numérica real
  F, C, FI, CO son numéricas enteras
Algoritmo:
  ** Carga de la matriz DHONT
  para F de 1 a NGRUP hacer
```

```
    para C de 1 a NESCAN hacer
      DHONT(F,C) ← VOTOS(F) / C
    finpara
  finpara
  ** Inicialización del vector ESCAN
  para I de 1 a NGRUP hacer
    ESCAN(I) ← 0
  finpara
  ** Reparto de escaños
  para I de 1 a NESCAN hacer
    MAXIMO ← -1
    ** Obtención de un máximo
    para F de 1 a NGRUP hacer
      para C de 1 a NESCAN hacer
        si DHONT(F,C) > MAXIMO entonces
          MAXIMO ← DHONT(F,C); FI ← F; CO ← C
        finsi
       finpara
     finpara
     ESCAN(FI) ← ESCAN(FI) + 1
     DHONT(FI,CO) ← -1
   finpara
Finsubprograma
Subprograma IMPRIMIR LISTADO
Algoritmo:
  escribir CIR, CENSO, NESCAN, VEMI, PEMICEN, ABSTE,
    PABSCEN, VVALI, PVALI, VBLAN, PBLAN, VNULO, PNULO
  para I de 1 a NGRUP hacer
    escribir GRUPO(I), VOTOS(I), PVCEN(I), PVEMI(I), ESCAN(I)
  finpara
Finsubprograma
```

Codificación BASIC:

```
10 REM ELECCIONES
20 CLS
30 DIM DHONT(15,40), GRUPO$(15), VOTOS(15),
       PVCEN(15), PVEMI(15), ESCAN(15)
40 GOSUB 100 :  REM ENTRAR DATOS
50 GOSUB 270 :  REM ORDENAR GRUPOS
60 GOSUB 340 :  REM CALCULAR PORCENTAJES
70 GOSUB 450 :  REM REPARTIR ESCAÑOS
80 GOSUB 650 :  REM IMPRIMIR LISTADO
90 END
100 REM ENTRAR DATOS
110 INPUT "Nombre de la circunscripción: ", CIR$
120 INPUT "Censo electoral: ", CENSO
130 INPUT "Número de escaños: ", NESCAN
140 INPUT "Número de grupos presentados: ", NGRUP
150 VVALI = 0
160 FOR I = 1 TO NGRUP
170   PRINT "Nombre del grupo n.º "; I; ": ";
180   INPUT "", GRUPO$(I)
190   INPUT "Número de votos: ", VOTOS(I)
```

Técnicas de programación estructurada

11.0. INTRODUCCION

Las técnicas de desarrollo y diseño de programas, que se utilizan en la programación convencional, tienen inconvenientes, sobre todo a la hora de verificar y modificar un programa. En la actualidad, están adquiriendo gran importancia las técnicas de programación, cuyo objetivo principal es el de facilitar la comprensión del programa y, además, permiten, de forma rápida, las ampliaciones y modificaciones que surjan en el periodo de vida de un programa o una aplicación informática.

Como hemos visto en el capítulo anterior, una forma de simplificar los programas, haciendo más sencilla su lectura y mantenimiento, es utilizar la técnica del diseño descendente de programas (TOP-DOWN).

En los últimos años, la técnica más utilizada que sigue las directrices TOP-DOWN es la *programación estructurada*.

La programación estructurada fue desarrollada en sus principios por Dijkstra, al que siguieron una gran lista de nombres que realizaron tanto métodos de representación de programas, como diferentes lenguajes de programación adaptados a la aplicación directa de estas técnicas; entre ellos podemos citar a Chapin, Warnier, Jackson, Bertini y Tabourier como autores de sistemas de representación y a Niklaus Wirth, Dennis Ritchie y Kenneth Thompson como autores de lenguajes de programación estructurados.

Existen diversas definiciones de la programación estructurada que giran sobre el denominado *Teorema de la estructura*.

Teorema de la estructura: Se basa en el concepto de diagrama o programa propio que consiste en que toda estructura tenga un solo punto de entrada y un solo punto de salida.

Todo diagrama o programa propio, cualquiera que sea el trabajo que tenga que realizar, se puede hacer utilizando tres únicas estructuras de control que son la secuencia, alternativa y repetitiva.

En la programación convencional, se suele hacer un uso indiscriminado y sin control de las instrucciones de salto condicional e incondicional, lo cual produce cierta complejidad en la lectura y en las modificaciones de un programa. Eliminar estas dificultades es uno de los propósitos de la programación estructurada y por ello, en ocasiones, se ha definido como la técnica de la programación sin saltos condicionales e incondicionales. Esto no es rigurosamente cierto y por lo tanto no lo tomaremos como definición, sino como una norma general.

Como consecuencia del párrafo anterior, podemos indicar que todo programa estructurado puede ser leído de principio a fin sin interrupciones, en la secuencia normal de lectura.

Al mismo tiempo que se obtiene una mayor clarificación del programa por medio de estas técnicas, la puesta a punto del mismo es mucho más rápida, así como la confección de su documentación.

Los programadores en la fase de diseño realizan cada tarea en *módulos* o *bloques*, los cuales pueden estandarizar y así formar su propia biblioteca de programas para su utilización en sucesivas aplicaciones.

En los distintos departamentos de informática existentes no siempre se dispone de los mismos programadores, con respecto al tiempo que se pretende que dure una aplicación, por lo cual es de suma importancia que un programa realizado por una persona sea fácil de modificar y ampliar por otra. En este sentido, la programación estructurada ofrece muchas ventajas para lograr estos objetivos.

Un programa estructurado es:

—Fácil de leer y comprender.
—Fácil de codificar en una amplia gama de lenguajes y en diferentes sistemas.
—Fácil de mantener.
—Eficiente, aprovechando al máximo los recursos del computador.
—Modularizable.

11.1. ESTRUCTURAS BASICAS

Como se indica en el apartado anterior, el teorema de la estructura dice que toda acción se puede realizar utilizando tres estructuras básicas de control, la estructura secuencial, alternativa y repetitiva. Esta afirmación es cierta y demostrable aunque su demostración se sale de los objetivos de este libro, por lo tanto asumimos su cumplimiento.

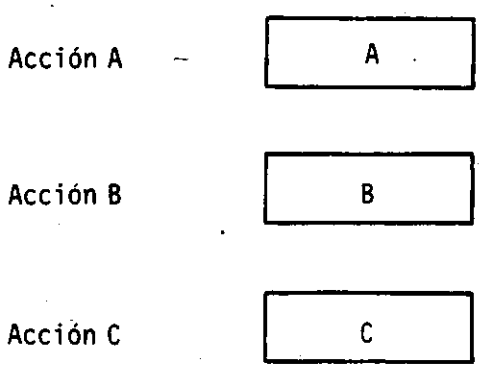
Para la representación gráfica de las estructuras utilizaremos el concepto de **acción** cuyo significado es totalmente general.

Una **acción** puede representar:

—Ninguna operación.
—Una operación sencilla. Como por ejemplo puede ser el movimiento de un valor de un campo a otro, una operación de salida, etc.
—Un proceso de cualquier tipo. Como por ejemplo una ordenación de datos o toda una tabla de decisión.

Con ello interpretaremos que una determinada acción representada en una de las tres estructuras puede estar compuesta por una o más estructuras en su interior.

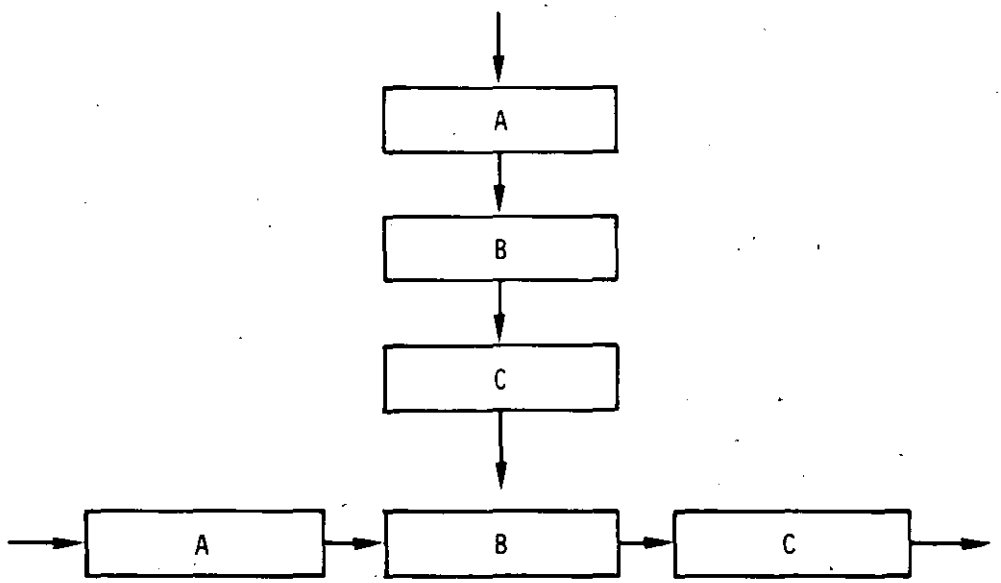
La notación utilizada para representar dichas acciones constará de rectángulos horizontales en cuyo interior pondremos letras mayúsculas.



11.1.1. ESTRUCTURA SECUENCIAL

Es una estructura con una entrada y una salida en la cual figuran una serie de acciones

cuya ejecución es lineal y en el orden que aparecen. A su vez todas las acciones tienen una única entrada y una única salida.

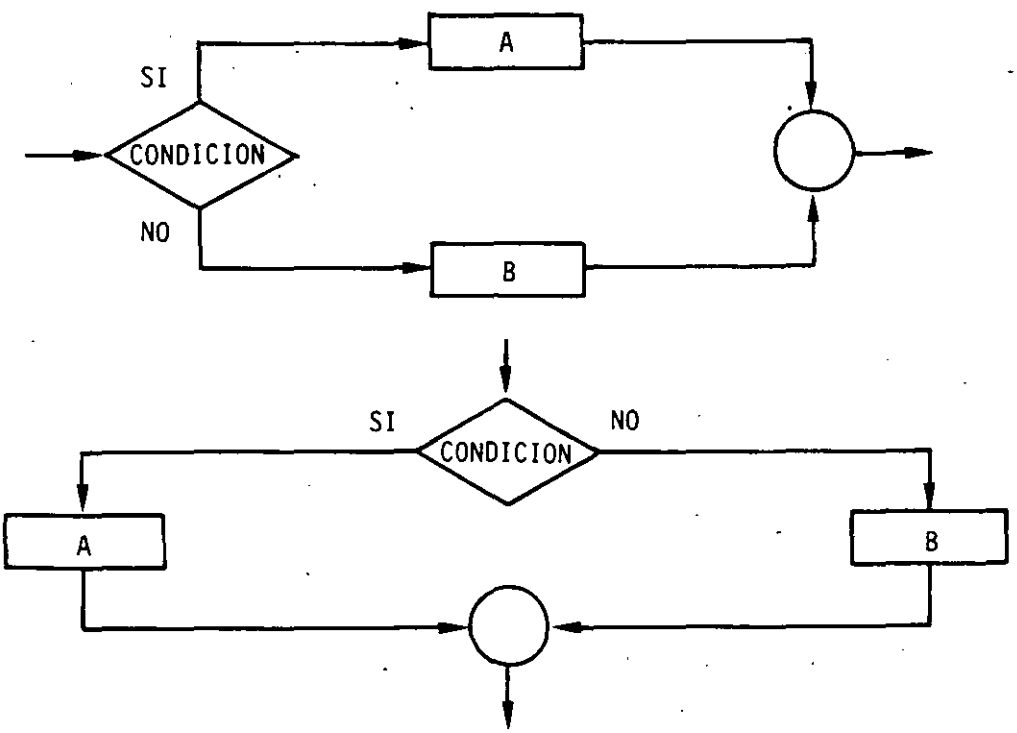


11.1.2. ESTRUCTURA ALTERNATIVA

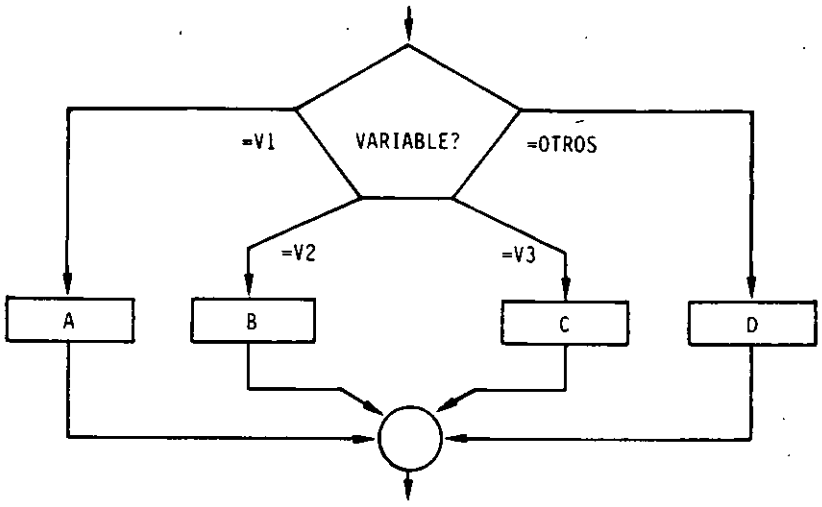
Es una estructura con una sola entrada y una sola salida en la cual se realiza una acción de entre varias según una condición o se realiza una acción según el cumplimiento o no de una determinada condición. Esta condición puede ser simple o compuesta.

Las estructuras alternativas pueden ser:

—De dos salidas. En la que una de ellas puede ser la acción nula.
—De tres o más salidas que también se llama múltiple.



Estructura alternativa doble



Estructura alternativa múltiple

11.1.3. ESTRUCTURA REPETITIVA

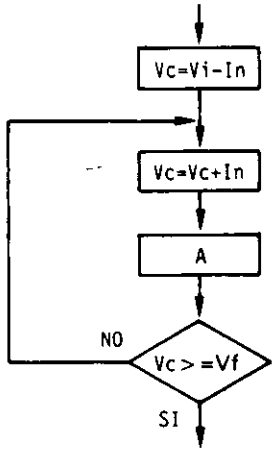
Es una estructura con una entrada y una salida en la cual se repite una acción un número determinado o indeterminado de veces, dependiendo en este caso del cumplimiento de una condición.

Las estructuras repetitivas pueden ser:

—Estructura para (FOR).
—Estructura mientras (WHILE).
—Estructura hasta (UNTIL).
—Estructura iterar (LOOP).

11.1.3.1. Estructura PARA (FOR)

En esta estructura se repite una acción un número fijo de veces representado normalmente por N. Es necesario para el control de la repetición utilizar una variable de control Vc y los valores que asignaremos a la misma inicialmente Vi y su correspondiente valor final Vf. El incremento de la variable de control Vc es normalmente 1 pero puede tomar otros valores positivos y negativos en cuyos casos es necesario indicarlo por medio de In.

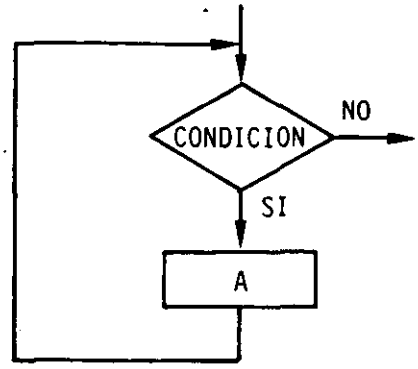


Donde N = parte entera $\left(\frac{Vf - Vi}{In}\right) + 1$.

11.1.3.2. Estructura MIENTRAS (WHILE)

En esta estructura se repite una acción mientras se cumpla la condición que controla el bucle.

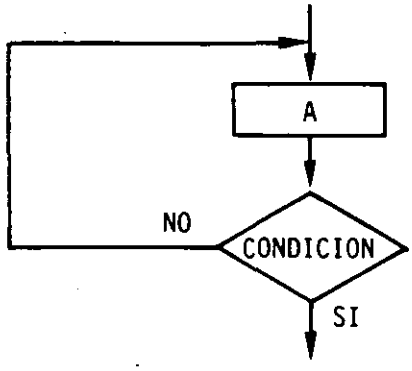
El número de repeticiones oscila entre 0 e infinito dependiendo de la evaluación de la condición, cuyos argumentos en los casos de repetición al menos una vez deberán modificarse dentro del bucle, pues de no ser así, el número de repeticiones será infinito y nos encontraremos en un bucle sin salida.



11.1.3.3. Estructura HASTA (UNTIL)

En esta estructura se repite una acción hasta que se cumpla la condición que controla el bucle, la cual se evalúa después de cada ejecución del mismo.

El número de repeticiones oscila entre 1 e infinito dependiendo de la evaluación de la condición, cuyos argumentos en los casos de repetición al menos dos veces deberán modificarse dentro del bucle, pues de no ser así, el número de repeticiones será infinito y nos encontraremos en un bucle sin salida.

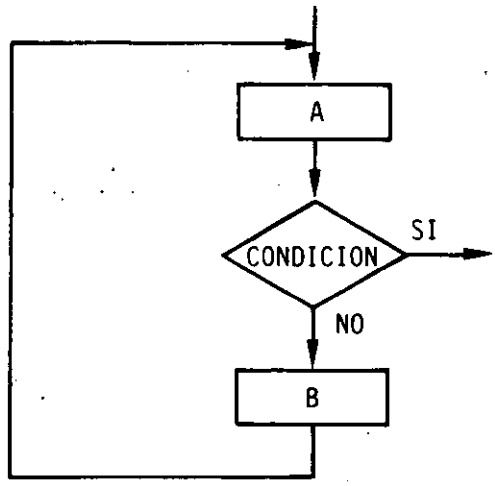


11.1.3.4. Estructura ITERAR (LOOP)

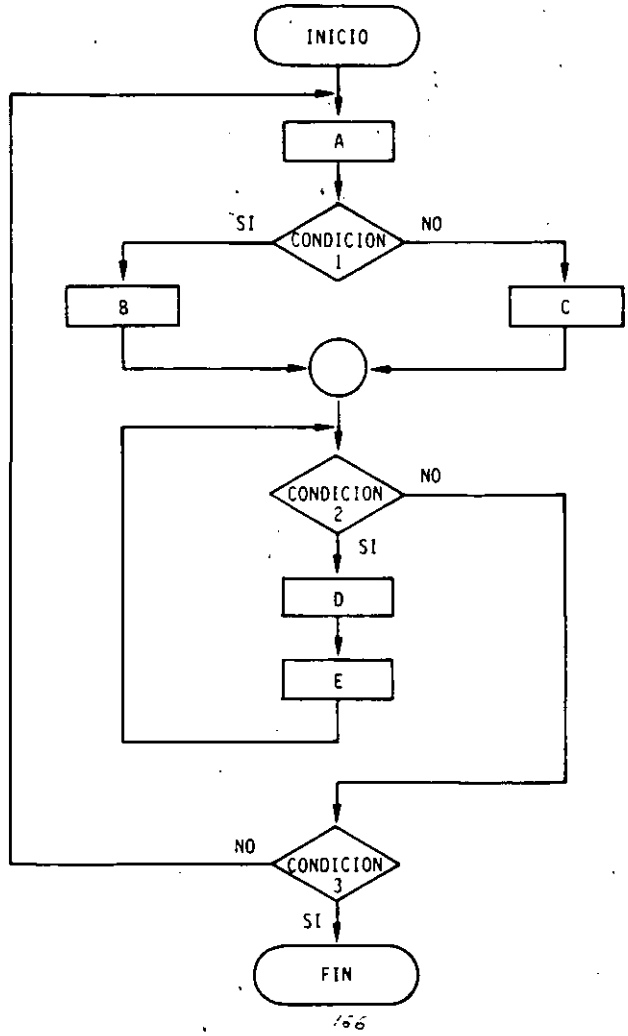
En esta estructura se repiten alternativamente dos acciones, evaluando la condición de salida entre ambas.

El número de repeticiones oscila, para la acción A entre 1 e infinito y para la acción B entre 0 e infinito, cumpliéndose que siempre se repite A una vez más que B.

Los bucles anteriores son casos particulares de éste.



Ejemplo de ordinograma estructurado.



11.2. METODO DE WARNIER

Se trata de un método para la representación de programas, cuyo resultado final se denomina diagrama de Warnier. En él podemos utilizar toda la terminología estudiada hasta ahora en lo que respecta a identificadores, constantes, variables, expresiones y operadores teniendo en cuenta que la característica fundamental en relación a todo lo anteriormente visto es la forma de diseñar el programa que será descendentemente, y la representación utilizada.

Este método se basa en el empleo de llaves de distintos tamaños que relacionan entre sí todas las tareas y operaciones.

Un diagrama de Warnier consta de:

—Estructuración de los datos de salida, representando los registros y archivos lógicos.
—Estructuración de los datos de entrada, representando los registros y archivos lógicos.
—Representación del proceso o algoritmo.

11.2.1. REPRESENTACION DE REGISTROS

Los registros se representan identificándolos por el nombre del archivo y descomponiéndolos sucesivamente en campos con sus nombres y dimensiones.

Ejemplos:

—Una variable V entera en la que establecemos una dimensión de 10 dígitos se representa:

```
V[(10)] [entera]
```

—Una variable X real en la que establecemos una dimensión de 5 dígitos en su parte entera y 2 en su parte decimal se representa:

```
X[(5.2)] [real]
```

—Una variable A alfanumérica en la que establecemos una dimensión de 10 caracteres se representa:

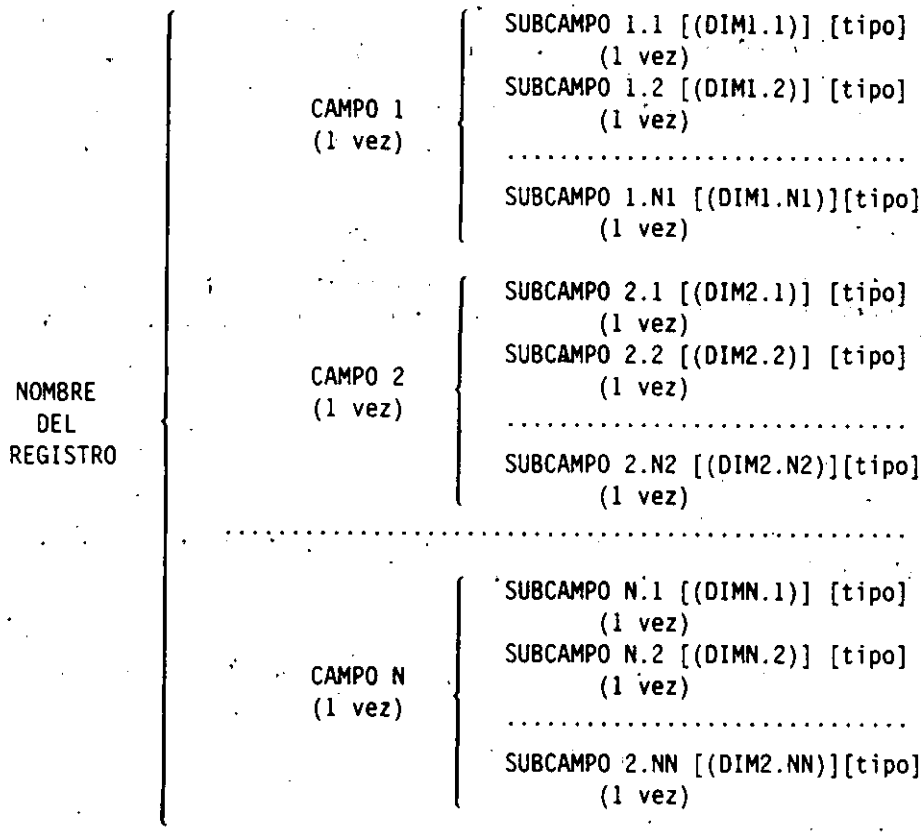
```
A[(10)] [alfanumérica]
```

En los casos que no sea necesario especificar la dimensión o el tipo se suprimen.
Los corchetes [ ] tienen el significado de opcional.

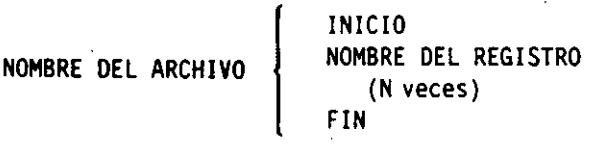
—Una constante numérica se representa por su valor y una alfanumérica entre comillas.
—Un registro compuesto de N campos (éstos serán variables numéricas o alfanuméricas con o sin dimensión) se representa:

```
                    { CAMPO 1 [(DIM1)] [tipo]
                    {           (1 vez)
                    { CAMPO 2 [(DIM2)] [tipo]
NOMBRE DEL REGISTRO {           (1 vez)
                    { ......................
                    { CAMPO N [(DIMN)] [tipo]
                    {           (1 vez).
```

—Un registro compuesto de N campos, los cuales a su vez están compuestos de N1, N2, ... NN subcampos, se representa:



—Un archivo compuesto de N registros se representa por:



De los puntos anteriores se pueden obtener todas las combinaciones posibles para la representación de archivos, que en cada caso vendrán impuestos por la definición del problema.

En los casos en que un campo, subcampo, etc., aparezca de forma consecutiva, no se repetirá en el diagrama. Se indicará acompañando al nombre del campo, subcampo, etc., con (N veces).

Estas representaciones son válidas tanto para los archivos lógicos de entrada como para los archivos lógicos de salida.

11.2.2. REPRESENTACION DEL PROCESO O ALGORITMO

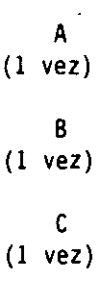
La representación del algoritmo se basa en los siguientes puntos:

—Un programa se representa por un solo diagrama en el cual se engloban todas las operaciones necesarias para la resolución del problema. Estas operaciones están colocadas secuencialmente a la derecha de una llave en cuya parte izquierda figura el nombre del programa.

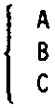
—En la parte superior de la llave anterior figurará el comentario INICIO.
—En la parte inferior figurará FIN.
—La forma de conectar con distintas páginas es a través de la palabra PROCESO seguida de un número, o un nombre que tenga relación con las operaciones que se realizan en la siguiente página. Estas palabras figurarán en el diagrama principal (diagrama que ocupa la primera página). En las siguientes figurará un diagrama sujeto a las mismas normas salvo que el nombre del programa será la palabra anteriormente citada. Las sucesivas conexiones se hacen de forma similar.
—Las estructuras tienen dos formas de representación, de las cuales utilizaremos la más sencilla.
—En los programas que no utilicen archivos, no es necesario definir con anterioridad la representación de los mismos, y en ningún caso las variables y constantes que se utilicen, debido a que éstas vendrán determinadas en el enunciado de cada problema.

11.2.2.1. Estructura secuencial

En esta estructura, las acciones se sitúan a la derecha de la llave y desde arriba hacia abajo según el orden de ejecución.

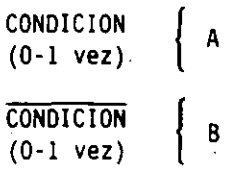


Como en la mayoría de los casos en una estructura secuencial no aparecen acciones repetidas de forma consecutiva la representaremos:

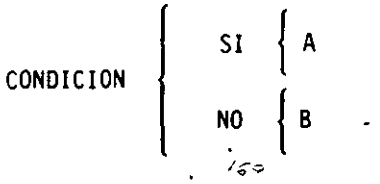


11.2.2.2. Estructura alternativa

En esta estructura se ejecutará una acción dependiendo de la condición.



También se puede representar de la forma:



En los casos en que una de las acciones sea nula se representa mediante un guión (-).

Por ejemplo, si en caso de cumplimiento de una condición se realiza una acción A y en caso contrario la acción nula (ninguna operación) la representación es:

```
CONDICION { SI { A
          { NO { -
```

La estructura alternativa múltiple se puede representar indistintamente por:

```
VARIABLE=V1  (0-1 vez) { A
VARIABLE=V2  (0-1 vez) { B
VARIABLE=V3  (0-1 vez) { C
VARIABLE=OTROS (0-1 vez) { D
```

O también:

```
VARIABLE { =V1     { A
         { =V2     { B
         { =V3     { C
         { =OTROS  { D
```

11.2.2.3. Estructura repetitiva

Como hemos visto anteriormente, esta estructura puede ser de varias formas cuya representación en este método es la siguiente:

11.2.2.3.1. Estructura PARA (FOR)

En esta estructura podemos indicar el número de repeticiones por N o a través de una variable de control, indicando sus valores inicial y final así como el incremento cuando éste sea distinto de 1.

```
PROCESO (N veces)              { A
PROCESO (Vc=Vi,Vf[,In])        { A
```

11.2.2.3.2. Estructura MIENTRAS (WHILE)

En este caso la acción se repite mientras se cumpla la condición que se evaluará siempre antes de cada repetición.

```
PROCESO
(mientras CONDICION)  { A
```

11.2.2.3.3. Estructura HASTA (UNTIL)

En este caso la acción se repite hasta que se cumpla la condición que se evaluará siempre después de cada repetición.

```
PROCESO
(hasta CONDICION)  { A
```

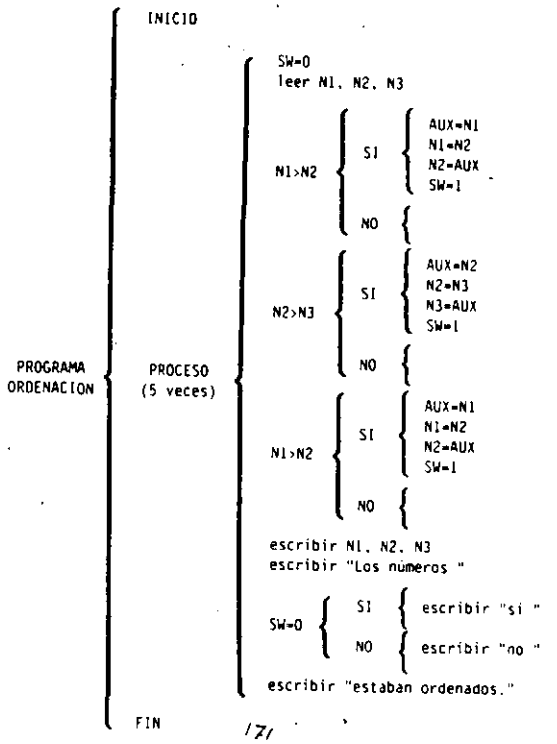
11.2.2.3.4. Estructura ITERAR (LOOP)

En este caso se repiten dos acciones alternativamente hasta que se cumple la condición que se evaluará siempre entre ambas acciones.

```
PROCESO  { A
         { (hasta CONDICION)
         { B
```

Ejemplo:

Dibujar un diagrama de Warnier representando un programa que lea 5 veces 3 números y los imprima ordenados ascendentemente indicando en cada ordenación si los números fueron introducidos ordenados o no.



/Z/

11.3. METODO PSEUDOCODIFICADO DE PROGRAMACION ESTRUCTURADA (PSEUDOCODIGO)

Se trata de un método para la representación de programas cuyo resultado final se denomina pseudocódigo.

Se basa en el empleo de corchetes ([) de distintos tamaños que relacionan entre sí todas las tareas y operaciones.

Al igual que en el método de Warnier, es válido todo lo estudiado anteriormente salvo la forma de representación.

Un pseudocódigo estructurado consta de:

—Estructuración de los datos de salida, representando los registros y archivos lógicos.
—Estructuración de los datos de entrada, representando los registros y archivos lógicos.
—Representación del proceso o algoritmo.

11.3.1. REPRESENTACION DE REGISTROS

Se representan de forma similar que por el método Warnier cambiando las llaves ({) por corchetes ([).

11.3.2. REPRESENTACION DEL PROCESO O ALGORITMO

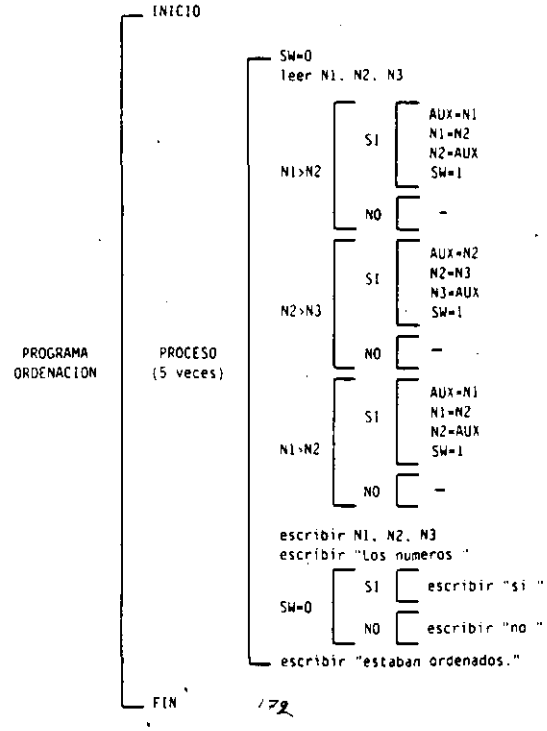
El proceso o algoritmo también se representa de igual forma que por Warnier cambiando las llaves ({) por corchetes ([).

La representación de estos pseudocódigos admiten la estructura de las instrucciones en castellano o en inglés (lo cual agiliza su codificación). Es decir, podemos poner por ejemplo: leer, escribir, inicio, etc., o read, write, start, etc.

Todas las estructuras básicas se representan de forma similar.

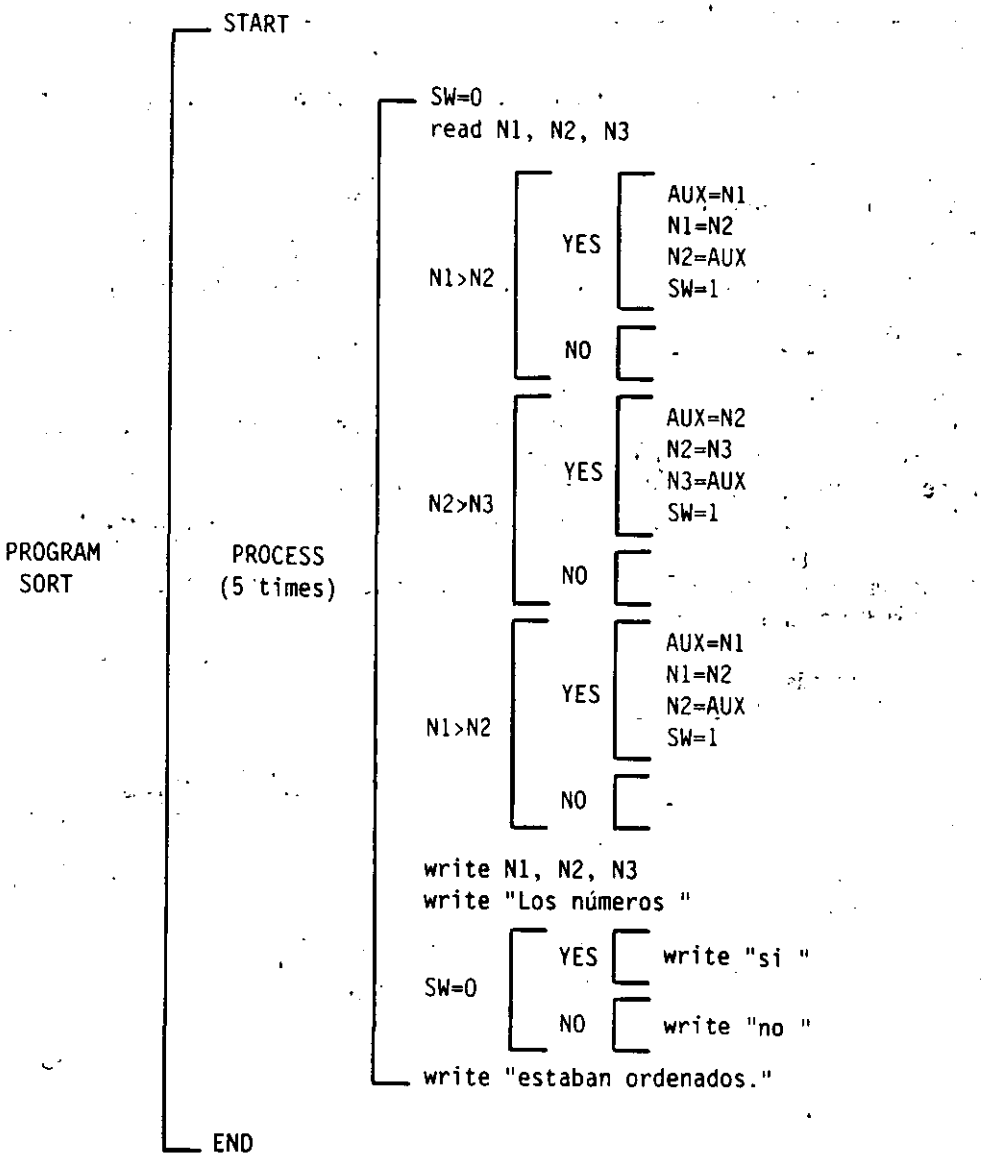
Ejemplo:

Dibujar un pseudocódigo en castellano para un programa que lea 5 veces 3 números y los imprima ordenados ascendentemente indicando en cada ordenación si los números fueron introducidos ordenados o no.



Ejemplo:

Dibujar un pseudocódigo en inglés para un programa que lea 5 veces 3 números y los imprima ordenados ascendentemente indicando en cada ordenación si los números fueron introducidos ordenados o no.



11.4. METODO DE JACKSON

Se trata de un método de representación de programas en forma de árbol denominado diagrama arborescente de Jackson. En él, al igual que en los métodos anteriores, se puede utilizar la terminología común a todos los métodos de representación.

Un diagrama de Jackson consta de:

—Definición detallada de los datos de entrada y salida incluyendo los archivos lógicos utilizados.

—Representación del proceso o algoritmo.

La simbología utilizada se basa en el empleo de rectángulos horizontales, que pueden presentar los siguientes aspectos:

La lectura del diagrama se hace recorriendo el árbol en preorden (RID), lo que supone realizar:

—Situarse en la raíz (R).
—Recorrer el subárbol izquierdo (I).
—Recorrer el subárbol derecho (D).

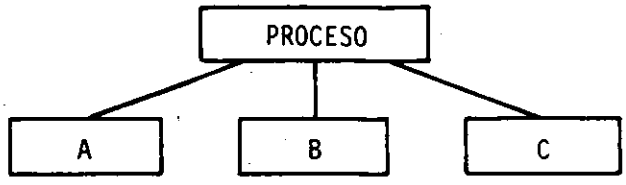
Cada subárbol se recorre igualmente en preorden hasta llegar a las hojas o nodos terminales del árbol.

La representación del algoritmo se basa en los siguientes puntos:

—Un programa se representa por un solo diagrama en el que se incluyen todas las operaciones a realizar para la resolución del problema. La forma de conectar una página con la siguiente es similar a los métodos anteriores, es decir, mediante la palabra PROCESO seguida de un número o un nombre encerrado en un rectángulo.
—Todo diagrama comienza con un rectángulo en cuyo interior figura el nombre del programa.
—Para conseguir una clara interpretación del diagrama es necesario realizar el mismo de forma lo más simétrica posible.

11.4.1. ESTRUCTURA SECUENCIAL

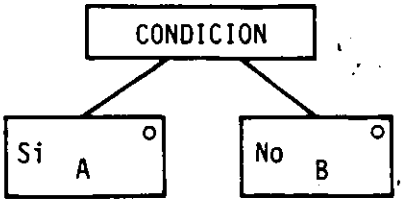
En esta estructura se ejecutan las acciones A, B y C de izquierda a derecha.



11.4.2. ESTRUCTURA ALTERNATIVA

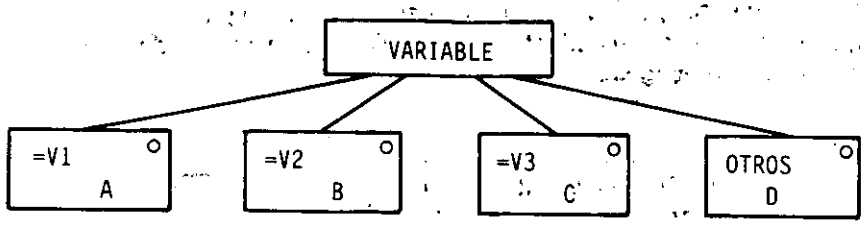
Se ejecuta una acción entre varias según la evaluación de una condición.

Estructura alternativa doble:



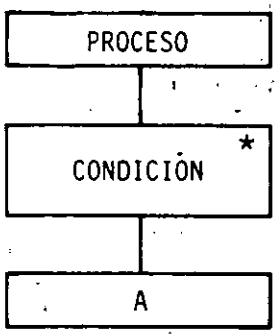
Si B es la acción nula, se escribe un guión (–).

Estructura alternativa múltiple:



11.4.3. ESTRUCTURA REPETITIVA

Se repite una acción dependiendo de una condición de fin de repetición.



11.4.3.1. Estructura PARA (FOR)

La CONDICION puede ser:

—N veces.
—Vc=Vi,Vf [,In]

11.4.3.2. Estructura MIENTRAS (WHILE)

Se escribirá:

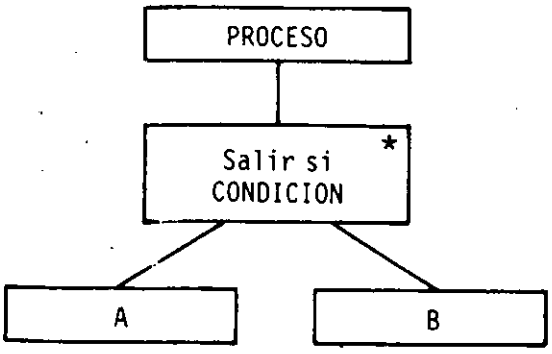
—mientras CONDICION.

11.4.3.3. Estructura HASTA (UNTIL)

Se escribirá:

—hasta CONDICION.

11.4.3.4. Estructura ITERAR (LOOP)



Ejemplo:

Dibujar un diagrama arborescente de Jackson que represente un programa que lea 5 veces 3 números y los imprima ordenados ascendentemente, indicando en cada ordenación si los números fueron introducidos ordenados o no.