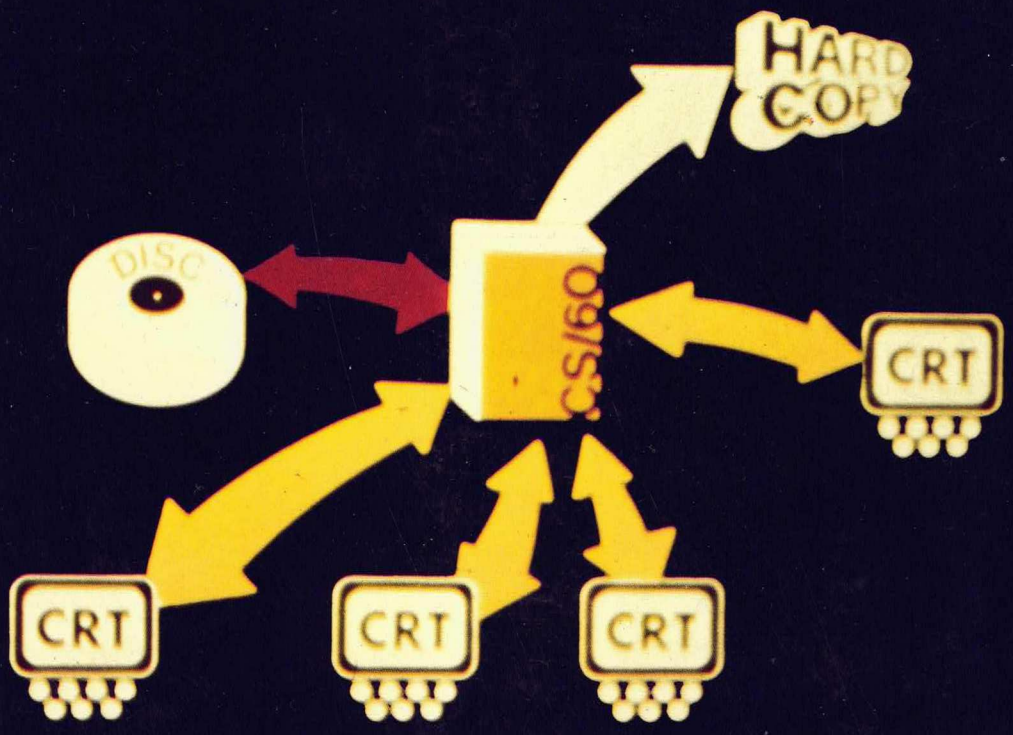
# PROGRAMACION DE ORDENADORES

## LEYES DE CONSTRUCCION DE PROGRAMAS.




EDITORIAL tesys

TOMO 9

El presente tratado de **Programación de Ordenadores** está compuesto por un conjunto de temas que constituyen la estructura imprescindible para todo aquel que desee un amplio conocimiento del campo de la Programación.

Dichos temas vienen englobados en volúmenes diversos que a continuación se detallan; constituyéndose por separado como unidades monográficas susceptibles de un estudio particular, al margen del conocimiento conjunto y global de todos ellos, como compendio completo para el logro final de programar.

La agrupación por tomos se desarrolla de la siguiente forma:

Tomo 1. **Estructura Básica del Proceso de datos.**

Tomo 2. **Estructura Básica del Proceso de datos.**

Tomo 3. **Estructura Básica del Proceso de datos.**

Tomo 4. **Estructura Básica del Proceso de datos.**

Tomo 5. **Estructura Básica del Proceso de datos.**

Tomo 6. **Lenguaje Basic**

Tomo 7. **Lenguaje Cobol**

Tomo 8. **Lenguaje Cobol**

Tomo 9. **Leyes de Construcción de programas.**

Tomo 10. **Diccionario de Proceso de datos**

El tratamiento de los temas comprende tanto los aspectos teóricos como las aplicaciones al diseño de estructuras básicas de posible uso inmediato, reagrupándose en los siguientes conjuntos específicos que por sí mismos pasan a ser unidades independientes.

Son los siguientes:

*Grupo I:* Tomos 1, 2, 3, 4 y 5. **(Estructura Básica del Proceso de Datos).**

*Grupo II:* Tomo 6. **(Lenguaje Basic).**

*Grupo III:* Tomos 7 y 8. **(Lenguaje Cobol).**

*Grupo IV:* Tomo 9. **(Leyes de Construcción de Programas).**

# PROGRAMACION DE ORDENADORES

## LEYES DE CONSTRUCCION DE PROGRAMAS

# PROGRAMACION DE ORDENADORES

## LEYES DE CONSTRUCCION DE PROGRAMAS

AUTOR

**JOSE LUIS MORALES PASCUAL**
Licenciado en Informática

COLABORADORES

**CARLOS ANTONIO MARTINEZ TREVIN**
Ingeniero de Telecomunicación

**ANDRES J. ERNESTO RUANA GIRONES**
Ingeniero Industrial

BARCELONA (ESPAÑA)

© E.C.C. - Barcelona, 1981

Editado por:
EDITORIAL TESYS - Bruch, 20 - Barcelona-10 (España), 1981

Primera edición: Mayo, 1981

Depósito Legal: B. 25.869-1980 (IX)

ISBN: 84-85838-00-9 (Obra completa)
ISBN: 84-85838-09-2 (Tomo 9)

Impreso por:
GRAFOS, Arte sobre papel.
Paseo Carlos I, 157 - Barcelona-13

Impreso en España - Printed in Spain

# CONTENIDO


Introducción ..... 9

1. Utilización del ordenador ..... 13

2. Fases de la programación ..... 23

3. Construcción de algoritmos y programas ..... 41

4. Diseño de un programa ..... 63

5. Codificación ..... 93

6. Pruebas de un programa ..... 119

7. Ejemplos de programas ..... 135

Soluciones de los ejercicios ..... 153

Indice general ..... 155

# INTRODUCCION

El objetivo del presente texto, es afianzar todos los conocimientos adquiridos hasta ahora y ponerlos en práctica mediante unas técnicas de diseño de programas. La forma de trabajo que se va a presentar se basa en un deseo común de todas las personas relacionadas con la informática de estandarizar o unificar la construcción de programas. También se desea la unificación en el desarrollo de lenguajes y en la construcción de ordenadores. La razón es simplemente la de facilitar la compatibilidad entre los distintos sistemas existentes.

Existen varias razones para la utilización de un método en el trabajo. Un diseño de un programa no es casi nunca resuelto en una tarde que el programador se ponga a pensar sobre ello, de modo que dejar a la intuición un trabajo que va a hacerse en varios días no es un método eficaz.

Corregir un programa es una tarea larga para el programador. Si éste hiciese este paso a la ligera, el ordenador se encontraría con muchas dificultades para hacer uso de ese programa. Estas trabas suponen una pérdida de tiempo, y en consecuencia, de dinero, que podría haberse evitado con una mínima atención del programador al corregir.

Vamos a adelantar una regla importante en la organización de la construcción de programas, aunque se desarrollará ampliamente a lo largo del texto.

La idea es estudiar o desarrollar un programa por módulos, cada uno de ellos con unas funciones concretas y diferentes que luego se enlazarán para hacer el programa total. Naturalmente cada módulo se puede desarrollar por separado y completamente. De este modo, el trabajo es más fácilmente llevadero porque se trabaja siempre con partes de programa y no con el programa entero.

Revisaremos también las etapas que deben seguirse en la resolución de un problema por ordenador que de todos modos deben conocerse del estudio precedente. Esto nos servirá para refrescar la memoria y para conducirnos al tema principal que es la programación.

Este tema se desarrollará completamente para conocer todos los detalles de la confección de un programa. Nos basaremos en temas vistos anteriormente que han debido ser asimilados por el lector. En dichos temas se estudian los elementos de los que dispone un programa para construirse, esto es, las estructuras de datos que maneja y los tipos de instrucciones que puede ejecutar.

Como comentario adicional diremos que en la medida de lo posible nos alejaremos de un lenguaje particular aunque para el desarrollo de los ejemplos, necesarios para la compresión del tema, será necesario elegir algún lenguaje de los estudiados. El BASIC es el más apto para el aprendizaje y es el que se usará.

Una vez hecha esta aclaración seguimos comentando el contenido del texto. Después de conocer los instrumentos pasaremos a desarrollar el modo en que un programa se introduce en el ordenador y qué hace la máquina con él. Ahora no vamos a entretenernos en cómo se almacena en memoria, cómo fluye la información, ya sean datos o instrucciones, de un módulo a otro del ordenador, sino que veremos qué transformaciones deben hacerse a un programa para ser ejecutable por la máquina en su lenguaje interno binario.

Veremos también cuáles son las causas de errores que puede introducir la máquina por su modo de almacenar los datos. Esto es, que no tenemos que contentarnos con que un programa aparezca sin errores después de haberse compilado, sino que el resultado sea exactamente el que queríamos obtener y no otro, que aunque perfectamente lógico no sea el que estábamos buscando. Veremos todos los mecanismos que tiene un programador para modificar un programa, haciéndolo más largo sin duda, pero éste será un inconveniente que se convierte en una ventaja, ya que nos permitirá modificar los errores, o al menos averiguar dónde están para poder corregirlos.

Una vez hayamos visto todas las ayudas en la puesta en marcha del programa se revisarán uno a uno los casos que vamos a encontrarnos en los problemas reales. Será ir presentando situaciones, ir viendo qué estructura de programación soluciona este caso. Veremos que todos los casos que se nos pueden presentar pueden reducirse a unos pocos y que sabiendo programar éstos, y uniéndolos adecuadamente podremos construir programas completos.

Presentaremos, después de conocer las estructuras básicas, distintos problemas que proporcionan mucha información de cómo funciona un algoritmo y que a la vez serán convenientes para crear una pequeña biblioteca de programas útiles, la cual puede usarse en equipos de informática que no dispongan de dicha biblioteca. En cualquier caso, el haber pensado en la solución de esos problemas será de gran ayuda para el lector, favoreciendo el esquema lógico de pensamiento cuando le toque diseñar un programa de ordenador.

# 1.

# UTILIZACION DEL ORDENADOR

## CUADRO RESUMEN DE LA LECCION

| | |
|---|---|
| Utilización del Ordenador | – Trabajos de creación |
| | – Ayudas a la educación |
| | – Gestión de datos y preparación de informes comerciales |
| | – Ayuda a la administración del Estado |
| | – Cálculo científico y técnico |
| | – Control de procesos |
| | – Inteligencia artificial |

Los ordenadores se pueden encontrar en cualquier actividad de la vida cotidiana. Puesto que las actividades son variables, también los tipos de ordenador que se usen para cada una de ellas serán distintos, en unos interesará la velocidad de cálculo, en otros, que tengan grandes memorias donde se puedan guardar y tratar gran cantidad de datos, y en otros interesará que posean una combinación de estas características.

El ordenador puede manejar datos en mucha mayor cantidad y rapidez que los humanos, puede realizar operaciones aritméticas a velocidades nunca alcanzables por el hombre. Para ello ha de ser programado para que pueda ejecutar su tarea. Esto significa que hay que decirle, sin ninguna ambigüedad, todo lo que debe hacer con los datos para obtener los resultados que se desean.

El hombre siempre se rodea de las herramientas que pueden facilitarle su trabajo, y el ordenador es una herramienta indispensable en la actual civilización.

## 1.1. TRABAJOS DE CREACION

En el campo estrictamente de creación existen trabajos realizados por el ordenador: libros de poesías, cuadros, figuras, música, etc., compuestos por la máquina.

Se está hablando de obras completas hechas por una máquina, no de las ayudas a la composición musical o artística que puedan hacer, ni de esas copias de cuadros, personajes famosos o dibujos animados. En este último caso, el

trabajo del ordenador es mera copia y la creación es de quien realizó la obra o el programa. Al ordenador sólo se le han dado las reglas para formar palabras o frases, la frecuencia de aparición de las palabras en un escrito, etc. Así pues,el ordenador ha llegado a tareas que parecían exclusivas de los hombres.

## 1.2. AYUDAS A LA EDUCACION

Siguiendo con las tareas de tipo social, se encuentran las ayudas en la educación. Desde preparar programas que hagan las estadísticas de los resultados obtenidos por los alumnos en una clase, hasta confeccionar el material que constituye la información para la enseñanza y evaluar el aprovechamiento de ésta.

La estadística de los resultados es de gran ayuda para el profesor y pedagogo. Se preparan unas pruebas objetivas en forma de test y se pasan a los alumnos para su contestación. Estas contestaciones marcadas por los alumnos, habitualmente sobre hojas de lectura óptica, constituyen los datos de entrada del programa.

El ordenador se encarga de manejar toda esa información diciendo los alumnos que han acertado o fallado en cada una de las preguntas, cuál ha sido el porcentaje de alumnos que no han contestado alguna pregunta, cuál el porcentaje de alumnos que marcaron una respuesta errónea, cuál es la nota de cada alumno, su posición en cuanto al resto de la clase, cuáles son los porcentajes respecto al total de la clase, de cada respuesta errónea que se marcó, y cualquier dato más que se quiera saber y pueda obtenerse de las contestaciones de los alumnos.

La ventaja inmediata que se le ve al sistema es la mecanización, y con ello, la facilidad de manejar grandes cantidades de exámenes, que serían de corrección muy pesada por parte del profesor si se tuviera que hacer la corrección a mano. Esto da una idea inmediata de la marcha del conjunto de la clase y exactamente de los puntos que han quedado más oscuros en las explicaciones. Evidentemente, si toda la clase no contesta a un grupo de preguntas de un mismo tema, significa que ese tema no se ha comprendido. Si además, todos los alumnos marcan la misma contestación errónea, es que ese tema no se explicó correctamente.

Algunas personas no consideran tantas ventajas al uso del ordenador en este campo, argumentando que no ahorra tiempo al profesor porque tiene que

preparar exámenes que sean realmente objetivos y que de ese modo, midan verdaderamente el aprovechamiento de los alumnos, y eso lleva mucho tiempo.

En cuanto a los programas de enseñanza, el ordenador presenta un tema al alumno mediante su pantalla de rayos catódicos (la forma más cómoda y habitual de diálogo con el ordenador, en este caso), y le va haciendo preguntas sobre el tema. Si esas preguntas están mal contestadas por el alumno, éste no puede pasar adelante, mostrándole el ordenador dónde se encuentra su error. En el momento en que el alumno ha encontrado la respuesta correcta, el ordenador le presenta una nueva pregunta.

La programación del ordenador no es demasiado complicada en ninguno de los dos casos expuestos. En el primer caso, el trabajo del ordenador consiste, ante todo, en organizar la estadística realizando muchas sumas y otras operaciones aritméticas sencillas. En el segundo caso, sólo debe representarse en la pantalla la información que constituye el tema. Se espera la respuesta del alumno, se analiza si es verdadera o falsa, continuando adelante o dando un mensaje de error, respectivamente.

## 1.3. GESTION DE DATOS Y PREPARACION DE INFORMES COMERCIALES

La gestión de datos y la preparación de informes comerciales es uno de los campos en que trabajan mayor número de ordenadores. Un ejemplo es el de los servicios bancarios en que el ordenador se encarga de añadir imposiciones, deducir reintegros, calcular intereses, transferir de una cuenta a otra, etc. Trabajo muy sencillo de cálculo, pero que se hace casi imposible por operadores humanos debido a la cantidad enorme de datos que deben manejar. Otros ejemplos son la gestión de nóminas en empresas con plantilla muy numerosa y la gestión de stocks en grandes almacenes. Son siempre operaciones muy sencillas pero que requieren mucho tiempo y con la posibilidad de incurrir fácilmente en errores por la gran cantidad de datos que se manejan.

## 1.4. AYUDA A LA ADMINISTRACION DEL ESTADO

El ordenador es de gran utilidad en la Administración, llevando al día los ficheros de los registros de propiedad, nacimiento, confección de recibos de los diversos servicios como agua, luz, teléfono, etc. El problema reside en manejar

muchos datos, de este modo el ordenador interesará que tenga gran capacidad de memoria, aunque sea «lento» en el cálculo. Los programas de manejo de estos datos deberán ser escritos en lenguajes que facilitan la escritura con formatos de salida que sean sencillos de interpretar por el usuario; además deben manejar con facilidad grandes cantidades de datos tanto en entrada como en salida.

## 1.5. CALCULO CIENTIFICO Y TECNICO

Otra de las muchas aplicaciones es la dedicada al cálculo científico y técnico. En este campo, las aplicaciones son muy variadas, desde la resolución de un problema matemático, hasta la ayuda a la navegación de un satélite, pasando por las ayudas en el diseño de máquinas o productos en ingeniería.

En resolución de problemas matemáticos el ordenador es de gran utilidad. Por ejemplo:

Calcular todos los números de tres cifras menores de 1.000 tal que la suma de los cubos de cada una de sus cifras coincida con dicho número. Este problema tiene una descripción matemática muy sencilla:

Sean *a, b* y *c* las tres cifras de un número. La cifra *a* es la de las centenas, la *b* la de las decenas y la *c* la de las unidades.

El número se puede descomponer en:

$$100 * a + 10 * b + c$$

La suma de los cubos de cada una de las cifras es:

$$a^3 + b^3 + c^3$$

La expresión matemática del problema es:

$$100 * a + 10 * b + c = a^3+b^3+c^3$$

Ir probando tríos de cifras para ver si cumple la relación es muy largo. Pero al ordenador no le cuesta demasiado trabajo comprobar si todos los tríos posibles cumplen la ecuación que expresa la condición propuesta.

Existen otros problemas muy laboriosos que necesitan el uso del ordenador para ser resueltos, porque forman parte de un estudio más amplio. Un ejemplo puede ser la resolución de sistemas de ecuaciones cuando el número de estas ecuaciones es muy grande.

A menudo, cálculos muy difíciles deben ser efectuados con rapidez. Por ejemplo, el cálculo de la trayectoria de un satélite, el cálculo del tiempo que deben tenerse conectados unos motores para cambiar de órbita, etc.

Otra aplicación del ordenador es, por ejemplo, ayudar al diseño de carrocerías de automóviles, o en el de altavoces o cajas acústicas. Se prepara en el ordenador un diseño de la carrocería o del altavoz, y se empiezan a ensayar distintas condiciones de trabajo. El ordenador puede simular lo que le ocurriría a esa carrocería si el viento le viniese en una dirección o en otra y con distintas fuerzas. De este modo, sin tener que fabricarlas se ensayan diferentes carrocerías y se construye la mejor. Igualmente los altavoces se pueden simular y ver cómo irían actuando frente a estímulos de diferentes tipos de música o de señales eléctricas que se les puedan aplicar; así, se va cambiando la forma o el material de que está hecho el altavoz hasta conseguir una respuesta satisfactoria.

## 1.6. CONTROL DE PROCESOS

En los sistemas utilizados para controlar la maquinaria de una fábrica se encuentran ordenadores que estudian las características de los productos que van saliendo de una determinada maquinaria. El ordenador comprueba si estas características son las deseables averiguando, en caso negativo, qué es lo que va mal en la producción, intentando corregirlo.

La misión del ordenador consiste en optimizar la producción, aumentando el rendimiento.

Un ejemplo de estos controladores de procesos puede verse en el control del consumo de carburante en el automóvil, encargándose de dosificar el combustible que llega en cada momento a los pistones, según sea la velocidad, las características del terreno, el peso que lleva el automóvil, etc.; siempre procurando que el automóvil gaste lo menos posible en cada condición.

## 1.7. INTELIGENCIAS ARTIFICIALES

En este capítulo encontramos también los sistemas robots, sean o no adiestrables. Esta distinción se hace en la medida en que unos sistemas son simples manipuladores que llevan piezas de un lugar a otro, las colocan en agujeros en su posición correcta o se encargan de ensamblar piezas entre sí,

pero siempre dedicados a una única tarea. Esto es, que no puede cambiársele el tipo de piezas ni el movimiento que realizan. Si se quieren usar para otra actividad deben ser programados otra vez.

Existen otros sistemas que evitan estas dificultades. Estos serían los sistemas robot adiestrables en los que se permite una preparación para cambiar de tarea. Son sistemas en los que *no* necesita cambiar el programa si cambian las condiciones de trabajo, sino que es el mismo robot quien lo hace. Si, por ejemplo, cambia la forma de la pieza que debe manejar, no es necesario indicarle características de la nueva pieza, sino que el robot la examina. Una vez conocidas las características de la nueva pieza el robot se reorganiza para poder cumplir la nueva actividad.

En realidad los programas que se aplican a los ordenadores se diseñan copiando los esquemas de actuación y aprendizaje del mismo cerebro humano. Desde luego, son programas muy difíciles de desarrollar y sus mejoras van parejas a los avances que se hacen en el estudio del cerebro y sus diversos mecanismos.

Para darse cuenta de la verdadera magnitud del problema, podría pensarse en cómo se le diría a una máquina que reconociera la letra «A» escrita a mano. Una persona no tiene dificultad en resolver ese problema. Cada persona hace una «A» diferente. La idea es que si se pudiera conocer cuáles son las reglas que usan los humanos para descubrir por qué eso es una «A», se podría construir un algoritmo a partir de esas reglas y dárselo al ordenador como un programa de reconocimiento de letras «A».

# 2.

# FASES DE LA PROGRAMACION

## CUADRO RESUMEN DE LA LECCION

| | |
|---|---|
| Fases de la Programación | – Identificación del problema<br>– Descripción matemática<br>– Análisis numérico<br>– Programación<br>– Prueba del programa<br>– Documentación<br>– Producción<br>– Interpretación de los resultados |

En la lección anterior se ha visto la diversidad de actividades que puede desarrollar un ordenador. Dar reglas para todas estas actividades resulta una tarea difícil, pero existen leyes comunes de las que pueden hacer uso los programadores. Esas reglas proporcionan ventajas de ordenación y facilidad en el diseño de los programas.

Esta lección irá dedicada a enmarcar en qué fase de la resolución de un problema mediante el ordenador se hace la programación. De este modo se recordarán cuáles son los pasos que han de seguirse hasta conseguir la resolución de un problema.

## 2.1. IDENTIFICACION DEL PROBLEMA Y DEFINICION DE OBJETIVOS

El primer paso consiste en conocer cuál es el problema y cuál es la meta a la que se quiere llegar.

Por ejemplo: Sea una empresa en la que se quiere calcular semanalmente la nómina de los empleados.

Una vez identificado el problema se pasa a definir los objetivos. Estos consisten en obtener una lista de todos los empleados, las horas que trabajan y cuánto tiene que cobrar cada uno.

Una vez estudiado el problema debe decidirse si es rentable la utilización del ordenador en la resolución de este problema.

Supongamos que hay tres empleados, que cobran 400 pesetas por hora trabajada. Cuando el problema está completamente definido se pasa a su descripción matemática.

## 2.2. DESCRIPCION MATEMATICA

Esta fase consiste en buscar algún modelo matemático, es decir, alguna serie de operaciones y cálculos encadenados que conduzcan a la solución. En el caso presentado se debe calcular cuál es el salario que se debe pagar a cada empleado. Esto se hace multiplicando las horas que ha trabajado durante la semana por 400.

## 2.3. ANALISIS NUMERICO

En este ejemplo no es necesario realizar un verdadero *análisis numérico* porque todas las operaciones descritas en el modelo matemático (operaciones aritméticas) son realizables directamente por la máquina.

En la etapa de *análisis del problema,* ya debe tenerse una idea sobre el tipo de ordenador y lenguaje a utilizar, que será el más conveniente al problema que vaya a resolverse. En este sencillo ejemplo cualquier ordenador y cualquier lenguaje pueden servir.

Existen problemas matemáticos que no pueden resolverse directamente por la unidad aritmética del ordenador. Para que el ordenador pueda resolverlos se emplean las técnicas de análisis numérico, dándole al ordenador un método de trabajo mediante el cuál se obtendrá el resultado deseado.

Un ejemplo es el del cálculo de la potencia n-ésima de un número cualquiera, A. La representación matemática es $A^n$, que significa multiplicar el número A por sí mismo un número n de veces.

$$A*A*A*A*A*A*A*A* \ldots \text{(n veces)}$$

Puesto que en algunas máquinas no se puede calcular la potencia n-ésima del número A, se debe explicar en el programa que debe multiplicarse el número A tantas veces como se lo indique el exponente.

## 2.4. PROGRAMACION

En esa fase se desarrolla un algoritmo para indicar a la máquina los pasos que debe seguir para resolver el problema.

Se distinguen en este apartado dos etapas. En primer lugar, la etapa de *construcción del algoritmo,* la cual debe estar muy avanzada después de la descripción matemática y del análisis numérico.

La siguiente etapa consiste en la *codificación* de ese algoritmo en un lenguaje que pueda ser interpretado por la máquina.

La descripción del algoritmo debe desembocar en la construcción de un organigrama o diagrama de flujo, y a partir de este organigrama se deberá comenzar a codificar el programa en el lenguaje elegido.

En el ejemplo descrito anteriormente, el algoritmo es el siguiente:

1.– Entrada de datos.
2.– Cálculo del salario.
3.– Salida de resultados.

El punto 1, *entrada de datos,* puede descomponerse a su vez en dos partes:

*a)* Lectura del nombre del empleado.
*b)* Lectura de las horas de entrada y salida diarias.

En el caso de que fueran necesarios más datos del trabajador podrían colocarse en el registro del empleado mediante algún programa anterior de creación del fichero. Esto puede ser una alternativa, pero puede diseñarse otra forma de entrada que sea más conveniente.

También se puede componer un fichero con los datos de todos los empleados. Para el programa se deja únicamente la lista de horas de entrada y salida diarias de cada empleado, pero sin estar encabezadas por su nombre. El ordenador distinguirá cuándo ha terminado un empleado y empieza el siguiente.

El paso dos, *cálculo del salario,* se resuelve mediante una serie de ecuaciones:

Horas trabajadas al día = Hora de salida − Hora de entrada,

que se representará simbólicamente para abreviar la ecuación por:

$$HTD = HS - HE$$

Horas trabajadas durante la semana = Horas trabajadas al día $\times$ 5 días (de lunes a viernes).

Cuya representación simbólica será:

HTS = HTD * 5

Salario = Horas trabajadas a la semana * 400

representado simbólicamente por:

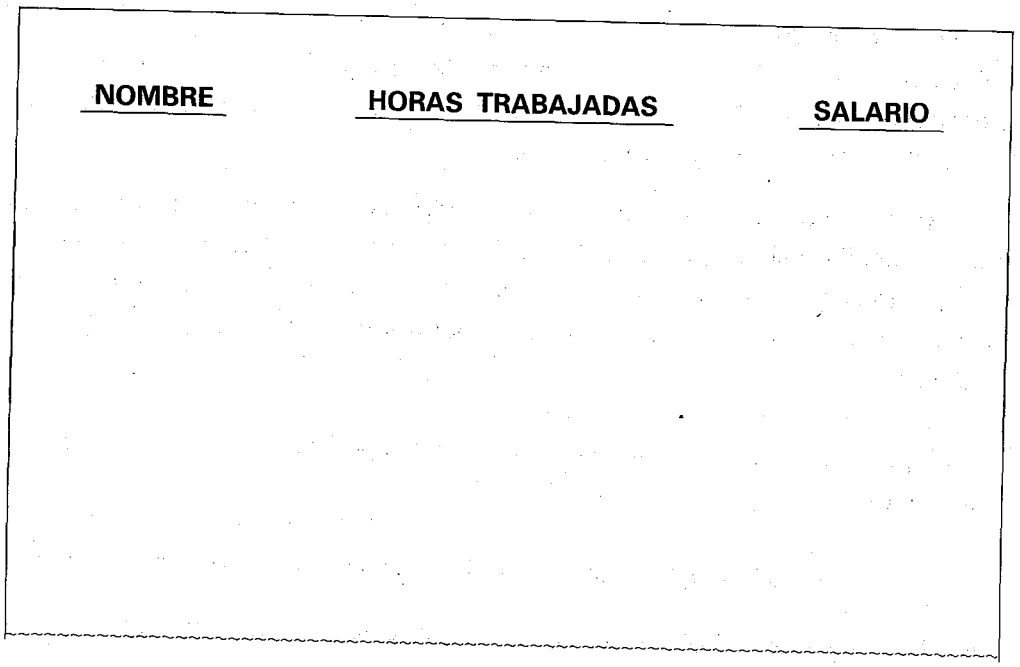
S = HTS * 400

La secuencia de operaciones en el paso dos es por lo tanto,

1. HTD = HS – HE
2. HTS = HTD * 5
3. S = HTS * 400

Este paso debe repetirse para cada uno de los empleados. En la *salida,* debe efectuarse la presentación de los resultados, lo cual se soluciona diseñando el formato de salida, es decir, qué información y cómo debe aparecer escrita.

Puede ponerse una cabecera de este tipo:




Debajo de cada columna irían apareciendo los datos obtenidos relativos a cada empleado: su nombre, el total de horas trabajadas y lo que hay que pagarle por su trabajo en una semana.

El algoritmo que corresponde a la solución de este problema es el siguiente:

```
EMPIEZA
    — ESCRIBIR cabecera.
    — REPETIR 3 veces:
        — ENTRAR el nombre del empleado (N$).
        — REPETIR 5 veces:
            — ENTRAR horas entrada y salida HE, HS.
            — HTD = HS – HE.
            — HTS = HTD + HTS.
        — HACER S = HTS * 4ØØ.
        — ESCRIBIR:
            — N$, HTS, S.
FINAL
```

La estructura y los símbolos empleados en este algoritmo se usarán a partir de ahora, en todos los algoritmos.

Todos los pasos del algoritmo están contenidos entre las palabras EMPIEZA y FINAL. El paso que se realiza siempre en primer lugar es el que está escrito debajo de la palabra EMPIEZA. El que se realiza en último lugar se escribe encima de la palabra FINAL.

Cada uno de los pasos o tareas del algoritmo está precedido por un guión. Cuando una tarea pueda dividirse en varias tareas más pequeñas, éstas se escribirán más a la derecha.

Se colocan dos puntos al final de la descripción de la tarea que tiene subdivisiones.

Cuando en alguna tarea se imprime algún literal contenido en la descripción, éste se coloca entre comillas.

En el algoritmo del ejemplo se tienen que realizar sólo dos tareas:

Escribir la cabecera.
Repetir tres veces:

Esta última tarea, «repetir tres veces:», tiene cuatro subdivisiones

*a*) Entrar el nombre del empleado (N$).
*b*) Repetir cinco veces:
*c*) Hacer S = HTS * 4ØØ.
*d*) Escribir:

Algunas de estas tareas también tienen subdivisiones.

La *b*), «Repetir cinco veces:», tiene tres subdivisiones:

Entrar horas de entrada y salida HE, HS.
HTD = HS − HE.
HTS = HTD + HTS.

La *d*), Escribir tiene una única subdivisión:

N$, HTS, S.

El algoritmo presentado es una de las posibles soluciones al problema.

En primer lugar se encuentra un proceso denominado *escritura de la cabecera*. Se coloca al principio antes de cualquier otra operación porque es una actividad del programa que será única. Si se colocara en el interior de algún bucle, la escritura de la cabecera se ejecutaría tantas veces como pasadas por el bucle hiciera el programa.

El siguiente paso es la ejecución de un bucle. Este se bucle se repite 3 veces, exactamente el número de empleados a los que se tiene que calcular el salario. En general, el bucle debe ampliarse al número total de empleados. La ventaja de esta situación es dar flexibilidad al programa, que será válido para cualquier número de empleados.

En el bucle de empleados, se lee el nombre de un empleado y seguidamente se entra en otro bucle, que se repetirá hasta 5 veces (los cinco días de la semana en que trabajan). En cada pasada por este segundo bucle se obtienen las horas de entrada y de salida de cada día, y en particular, para cada empleado. Dentro de este mismo bucle, y con los datos obtenidos en él, se realizan las operaciones destinadas a obtener el salario. En sucesivas pasadas del bucle se va obteniendo el total de horas trabajadas en la semana y al finalizar éste se está en condiciones de calcular el salario para ese trabajador.

La siguiente tarea del ordenador consiste en escribir los resultados obtenidos, como son: nombre del trabajador, horas trabajadas y salario.

Una vez realizada la impresión de todos estos resultados se repiten las mismas operaciones con el siguiente empleado.

Este proceso continúa hasta que finalice el tratamiento de los datos de todos los empleados. El organigrama de este problema se presenta en la Figura 2.1.

Una vez realizado el organigrama se debe codificar en un lenguaje de programación (por ejemplo en BASIC).

La codificación es un proceso muy sencillo pues se trata únicamente de traducir cada una de las partes del organigrama en la representación simbólica que corresponda al lenguaje usado.

Por ejemplo, veamos cómo se traduciría la expresión «leer el nombre del empleado». El programador debe asignar una variable del tipo alfanumérico al *nombre* (N$) y usar una instrucción de entrada de datos (de lectura). La traducción o codificación en BASIC es:

```
4Ø READ N$
```

donde 4Ø es el número de la instrucción, READ es la operación de lectura y N$ la variable donde se guarda el *nombre del empleado.*

De esta forma se siguen analizando y traduciendo los demás pasos del organigrama hasta conseguir la codificación completa.

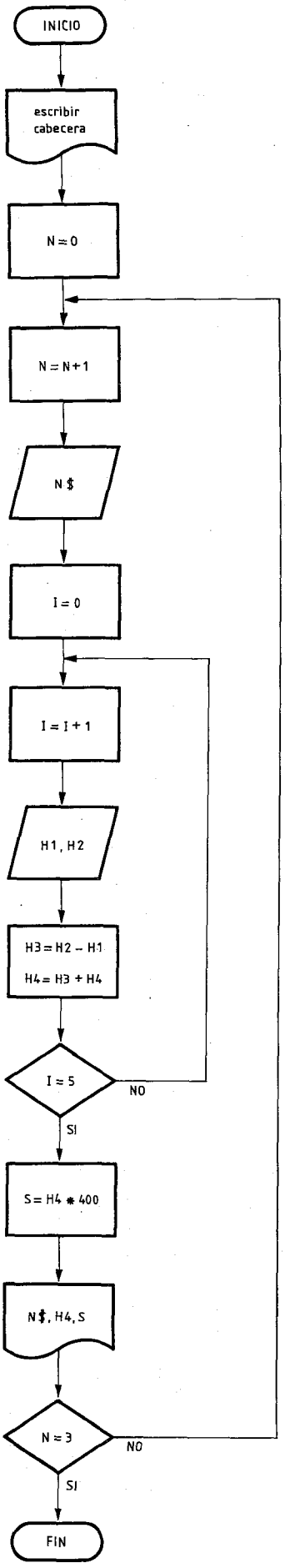



*Fig. 2.1. Cálculo de nóminas para 3 empleados.*

```
1Ø PRINT "NOMBRE", "HORAS TRABAJADAS", "SALARIO"
3Ø FOR N = 1 TO 3
4Ø READ N$
45 H4 = Ø
5Ø FOR I = 1 TO 5
6Ø READ H1, H2
7Ø H3 = H2 - H1
75 H4 = H3 + H4
8Ø NEXT I
9Ø S = H4 * 4ØØ
1ØØ PRINT N$, H4, S
11Ø NEXT N
12Ø STOP
13Ø DATA ...
14Ø END
```

En la tabla 2.1. se representa el nombre de las variables que se han usado en el programa. No se ha podido conservar el nombre dado en el algoritmo ya que en el lenguaje BASIC no se permite representar las variables con tantos caracteres.

| *Nombre en el algoritmo* | *Nombre en el programa* | *Tipo de variable* |
|---|---|---|
| Nombre | N$ | Alfanumérica |
| HE | H1 | Numérica |
| HS | H2 | " |
| HTD | H3 | " |
| HTS | H4 | " |
| S | S | " |
| | N | Numérica interna para el bucle |
| | I | Numérica interna para el bucle |

*Tabla 2.1.*

Además de las instrucciones estrictas de la codificación del programa, el programador incluye otras instrucciones de control para que el sistema operativo del ordenador ejecute el programa y tenga control sobre él.

Con estas instrucciones añadidas, el programa está listo para ser ejecutado.

## 2.5. PRUEBA DEL PROGRAMA

La fase denominada *prueba del programa (debugging)* consiste en corregir el programa.

En esta etapa se comprueba si el programa funciona y si calcula lo que se desea obtener.

Esta fase se efectúa con ayuda del ordenador, haciendo sucesivas pasadas hasta obtener un programa sin errores de resolución.

Los errores sintácticos pueden ser causados en el momento de la perforación del programa, o pueden ser fallos de construcción del organigrama, es decir, errores en la lógica del programa.

Por ejemplo, la máquina puede presentar un error de la siguiente forma:

```
SYNTAX ERROR AT LINE 4Ø
```

El programador busca la línea 4Ø del programa y lee:

```
4Ø RED N$
```

El error consiste en que la instrucción RED no pertenece a dicho lenguaje. Por lo tanto, el ordenador no puede ejecutar esa instrucción. La instrucción correcta es READ.

El programador añade la letra A y la instrucción queda

```
4Ø READ N$
```

Se vuelve a perforar esta instrucción y ahora sí será interpretada correctamente por la máquina.

Repasar el programa detenidamente antes de pasarlo por la máquina supone un ahorro importante de tiempo de máquina y de tiempo total de desarrollo del programa.

Una vez se tiene esta primera parte sintáctica corregida, se introducen en el ordenador unos datos para probar el funcionamiento del programa. A continuación se ejecuta el programa. Entonces, pueden producirse nuevos errores porque los cálculos están mal programados, por no darle valor inicial a una variable, o por cualquier motivo imprevisto.

Una buena técnica para evitar este inconveniente es haber resuelto el problema con la simple ayuda de una calculadora de bolsillo, siguiendo los pasos que indica el algoritmo. Una vez se ha probado el programa, deben compararse los resultados de la máquina con los obtenidos a mano, siempre que sea posible.

Para la prueba del programa en un determinado ordenador, se prepara un registro de datos por cada empleado (tres en total). En este registro se colocan los datos separados por comas. En primer lugar el nombre del empleado y seguidamente cinco parejas de números. El primer elemento de la pareja representa la hora de entrada y el segundo la de salida.

Registro 1: Juan, 7,15,7,15,7,15,7,15,7,15
Registro 2: Pedro, 8,12,8,15,Ø,Ø,8,15,8,15
Registro 3: Andrés, 5,8,5,8,5,8,5,8,Ø,Ø

Y los resultados obtenidos son los siguientes:

| NOMBRE | HORAS TRABAJADAS | SALARIO |
|---|---|---|
| JUAN | 4Ø | 16.000 |
| PEDRO | 25 | 10.000 |
| ANDRES | 12 | 4.800 |

## 2.6. DOCUMENTACION

En el momento que el programa está funcionando correctamente al programador sólo le queda documentarlo e introducir los datos reales para obtener los resultados.

La documentación de un programa consiste en añadirle todos los comentarios necesarios para un estudio posterior del programa o para hacer alguna modificación al mismo.

En programas pequeños la documentación no es estrictamente necesaria, porque se entiende perfectamente lo que realiza, pero conviene efectuarla porque la memoria del hombre es limitada, y puede olvidar en algún momento, la idea que tuvo para efectuar el cálculo. Con mayor motivo se debe realizar si es otra persona, ajena a la confección del programa, la que queda encargada de corregirlo.

Los comentarios que podían introducirse en este sencillo ejemplo serían los siguientes:

```
 5 REM **CALCULO DE NOMINAS PARA 3 EMPLEADOS**
 6 REM DATOS: NOMBRE EMPLEADO, HORA DE ENTRADA Y SALIDA.
 7 REM PARA CADA UNO DE LOS CINCO DIAS.
35 REM AQUI SE LEE EL NOMBRE DEL EMPLEADO.
55 REM AQUI SE LEEN LAS HORAS DE ENTRADA Y SALIDA.
65 REM CALCULO DE HORAS SEMANALES TRABAJADAS.
85 REM CALCULO DEL SALARIO.
95 REM ESCRITURA DE RESULTADOS.
```

Con lo que el programa completo con datos y comentarios quedaría de la forma siguiente:

```
 5 REM **CALCULO DE NOMINAS PARA 3 EMPLEADOS**
 6 REM DATOS: NOMBRE EMPLEADO, HORA DE ENTRADA Y SALIDA.
 7 REM PARA CADA UNO DE LOS CINCO DIAS.
1Ø PRINT "NOMBRE", "HORAS TRABAJADAS", "SALARIO".
3Ø FOR N = 1 TO 3
35 REM AQUI SE LEE EL NOMBRE DEL EMPLEADO.
```

```
4Ø  READ N$
45  H4 = Ø
5Ø  FOR I = 1 TO 5
55  REM AQUI SE LEEN LAS HORAS DE ENTRADA Y SALIDA.
6Ø  READ H1, H2
65  REM CALCULO DE HORAS SEMANALES TRABAJADAS.
7Ø  H3 = H2 - H1
75  H4 = H3 + H4
8Ø  NEXT I
85  REM CALCULO DEL SALARIO.
9Ø  S = H4*4ØØ
95  REM ESCRITURA RESULTADOS.
1ØØ PRINT N$, H4, S
11Ø NEXT N
12Ø STOP
13Ø DATA JUAN,7,15,7,15,7,15,7,15,7,15
132 DATA PEDRO,8,12,8,15,Ø,Ø,8,15,8,15
133 DATA ANDRES,5,8,5,8,5,8,5,8,Ø,Ø
14Ø END
```

## 2.7. PRODUCCION DEL PROGRAMA

En la etapa de producción o ejecución se le añaden los verdaderos datos con los que va a ejecutarse el programa.

En la mayoría de los casos el programa no se ejecuta una sola vez, sino que se ejecuta cada vez que se requieran sus cálculos. En nuestro ejemplo, el ordenador tendrá que usar cada semana el programa para realizar las nóminas.

## 2.8. INTERPRETACION DE LOS RESULTADOS

Donde puede verse más claramente el caso de la interpretación de resultados es en programas de cálculo científico. Saber la correlación existente entre los datos y los resultados, y ver cómo varían los resultados en función de los datos es una forma de entender el funcionamiento del sistema que se está estudiando.

En muchos casos la fase de interpretación no requiere gran esfuerzo. Por ejemplo, en el caso que se está resolviendo, cuando se leen los resultados se puede observar inmediatamente que cada línea corresponde a un trabajador, a la izquierda encontrará su nombre, a continuación las horas que ha trabajado a la semana y por último el salario.

De estos resultados puede conocerse quién es el empleado que ha faltado más veces, quién es el que ha trabajado menos horas, etc. De este modo puede conocerse la situación del personal de la empresa.

## EJERCICIOS

1. En la resolución de un problema, la creación del algoritmo corresponde al:

   *a*) Analista.
   *b*) Ordenador.
   *c*) Programador.

2. La definición de objetivos se relaciona con:

   *a*) Los datos de salida.
   *b*) Los datos de entrada.
   *c*) La construcción del programa.

3. El análisis numérico

   *a*) Es necesario siempre.
   *b*) Es un paso eventual.

4. El análisis numérico consiste en:

   *a*) Una forma ordenada de presentar al ordenador cálculos matemáticos.
   *b*) Analizar los resultados obtenidos por el programa.

5. Para la descripción matemática:

   *a*) Es necesario el ordenador.
   *b*) No es necesario el ordenador.

6. La programación se realiza teniendo en cuenta el lenguaje de programación empleado.

   *a*) Siempre, en cada una de sus fases.
   *b*) Sólo en la fase de codificación.
   *c*) Sólo en la fase de construcción del algoritmo.

7. El organigrama es

   *a*) La representación gráfica del algoritmo.
   *b*) Una descripción de la organización de la identificación del problema.

8. En el ejemplo presentado, las estructuras de la organización del programa que se han usado son:

   *a*) Bucles de entrada.
   *b*) Bucles de salida.
   *c*) Cálculos.
   *d*) Todas ellas.

9. Algún sinónimo cercano a codificación sería:

   *a*) Cálculo.
   *b*) Representación.
   *c*) Traducción.

10. El programa, una vez codificado, debería leerse:

    *a*) Antes de pasarlo al ordenador.
    *b*) Después de obtener los resultados de las pruebas y corregirlo.
    *c*) En ambas ocasiones.

11. Los comentarios en el programa:

    *a*) No deben usarse porque quitan espacio en la memoria.
    *b*) Deben usarse porque indican el funcionamiento del programa.
    *c*) No deben usarse porque presupone que los programadores van a entender el programa.

12. La interpretación de los resultados

    *a*) Sigue siempre a la etapa de producción.
    *b*) Es necesaria sólo en algunas ocasiones.

13. La interpretación de los resultados se utiliza:

    *a*) Para mejorar el algoritmo.
    *b*) Para conocer la relación de datos de entrada y salida.
    *c*) Para ambas funciones.

14. La producción significa:

    *a*) Obtener organigramas.
    *b*) Obtener programas.
    *c*) Obtener resultados.

# 3.

# CONSTRUCCION DE ALGORITMOS Y PROGRAMAS

## CUADRO RESUMEN DE LA LECCION

**Construcción de algoritmos y programas**

- Reglas
- Algoritmos de exploración de datos.
  - Secuencial.
  - Binaria.
  - Binaria con interpolación.
  - En árbol binario.
  - De condensación.
- Características de un programa.
- Criterios en la construcción de programas.
  - Criterio de normalización.
  - Criterio de realización.

Un ordenador es una máquina que maneja datos según las indicaciones de un programa para obtener resultados. En esta definición existen tres conceptos importantes: *el conjunto de datos, el programa* y *los resultados*. Ninguno de ellos tiene sentido sin los otros dos, de modo que para la confección de un programa eficiente hay que tener en cuenta los datos que va a manejar y los resultados.

La creación de un programa eficiente es el objetivo de esta lección.

Destinados a la consecución de este objetivo se desarrollarán las técnicas de construcción de algoritmos y cómo pueden organizarse los datos de entrada para mejorar el tiempo de respuesta del ordenador.

Se describirán a la vez tipos de problemas que pueden ser resueltos de diversas formas por medio de algoritmos.

A continuación se describirán las condiciones que debe tener un programa para que pueda adaptarse al conjunto de programas basados en la metodología de programación que se sigue. Se analizarán las estructuras de datos e instrucciones que permiten los lenguajes de programación que se utilizan.

## 3.1. REGLAS EN LA CONFECCION DE ALGORITMOS

Un algoritmo es un conjunto *finito* de reglas *ordenadas* y *concretas,* destinadas a la consecución de un resultado a partir de un dato o de un conjunto de ellos.

La primera regla se refiere al orden en que se ejecutan las instrucciones. El programa comprende una serie de instrucciones que se ejecutan en orden, una tras otra (secuencialmente).

El programador debe saber exactamente, cuando vaya a confeccionar un algoritmo, en qué momento se ejecutará cada instrucción, cuál se ejecutará antes y cuál después. Deberá decidir correctamente el flujo del programa, de modo que habiendo varias posibilidades se elija la más eficaz, es decir, la de ocupación mínima de memoria y del mínimo tiempo de proceso.

En segundo lugar, las reglas han de ser concretas y sin ambigüedades. Esto se debe a la forma de actuar del ordenador, ya que es una máquina que no puede decidir nada por sí misma y se le debe decir exactamente lo que debe hacer. Por lo tanto, no se puede desarrollar un algoritmo si no existe una colección de reglas concretas.

## 3.2. ALGORITMOS DE EXPLORACION DE DATOS

La exploración de datos consiste en elegir un dato entre todos los existentes en un conjunto.

Un algoritmo puede mejorarse si se cambia la forma de ordenar los datos que maneja. Se presentan tres algoritmos que mejoran paulatinamente el tiempo de proceso por una distinta configuración de los datos.

Los algoritmos son:

- algoritmo de exploración secuencial.
- algoritmo de exploración binario con interpolación en árbol binario.
- algoritmo de exploración por condensación.

Cada algoritmo supera al anterior, es decir, el algoritmo de condensación emplea menos tiempo de proceso que el de exploración en árbol binario y éste mejora al algoritmo secuencial. Esto significa que la primera solución encontrada no es siempre la mejor.

### 3.2.1. Exploración secuencial

La exploración secuencial consiste en recorrer en orden cada uno de los datos (secuencialmente).

*Ejemplo*

Supongamos que se tiene una lista de palabras almacenadas en la memoria. Como dato de entrada se utiliza una palabra cualquiera, que llamaremos CLAVE, y como salida, el ordenador debe dar la posición donde se encuentra la palabra que es igual a CLAVE dentro de la lista. Si CLAVE no pertenece a la lista, la posición de salida sería cero.

Para encontrar la posición que ocupa la palabra CLAVE dentro de la lista, en primer lugar se escoge la primera palabra de la lista (o la última) y se comprueba si coincide con CLAVE (palabra de entrada). Si coinciden ambas palabras, la salida es la posición 1 (o la última posición, N). En caso contrario, se continúa la búsqueda con la siguiente palabra de la lista. Esta comprobación se sigue realizando con las sucesivas palabras de la lista hasta que coincidan.

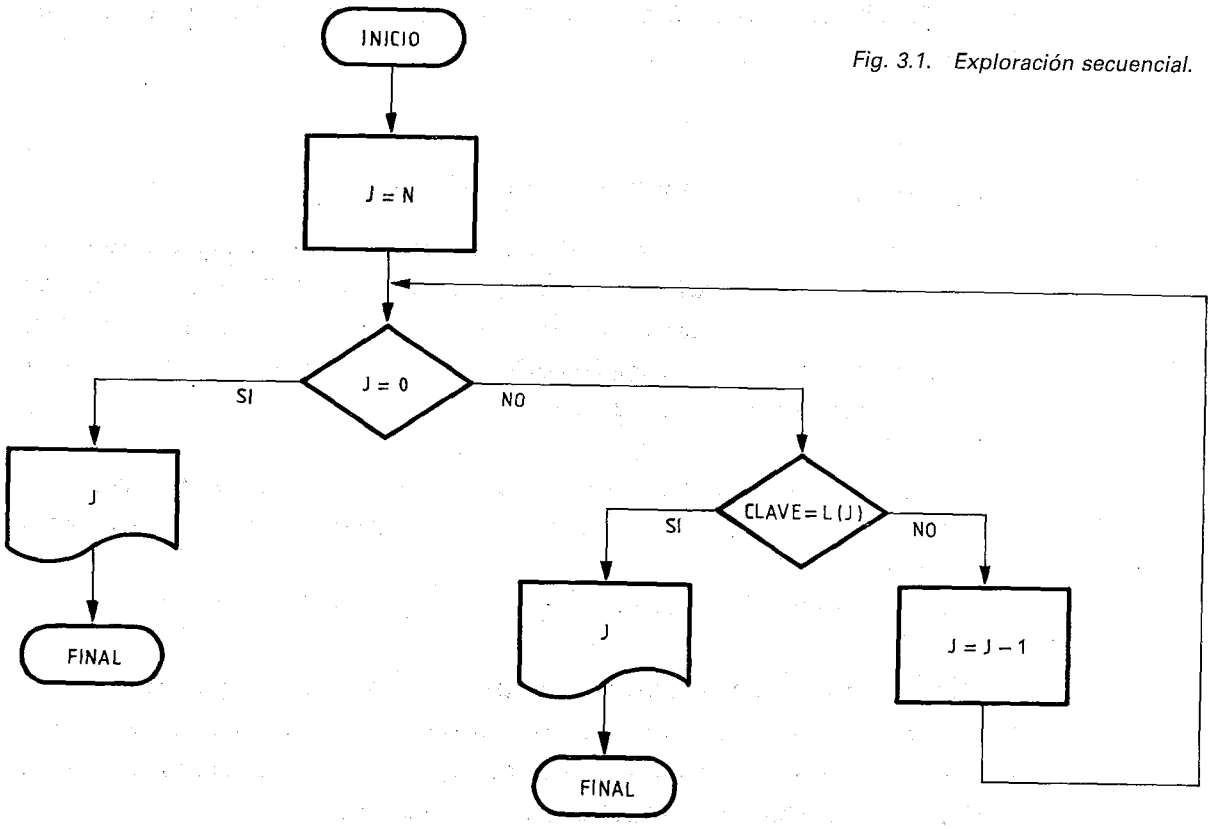



*Fig. 3.1. Exploración secuencial.*

En caso de haber revisado toda la lista sin haber obtenido éxito, la salida del programa sería cero.

El organigrama de esta primera solución se presenta en la figura 3.1., en la que:

N = n.° total de palabras almacenadas.
J = índice que indica la posición de una palabra.
CLAVE = palabra de entrada.
L(J) = palabra de la lista que ocupa la posición J.

Se puede comprobar que el algoritmo, si bien resuelve el problema, no es el más rápido en todos los casos. Observemos que si en todas las ocasiones CLAVE coincide con una palabra de los últimos lugares de la tabla, el algoritmo tendrá que recorrer toda la lista hasta dar con la solución. No obstante, para tablas cortas, el método es correcto.

Puede ocurrir que en lugar de tener en la memoria simples palabras almacenadas se tengan claves que abren registros de un fichero en el que hubiera de hacerse alguna transformación.

### 3.2.2. Algoritmo de exploración binaria

Teniendo las palabras ordenadas alfabéticamente, una exploración que emplea menos tiempo que la secuencial es la que utiliza el *algoritmo de exploración binaria:* en cada paso divide la tabla (o lista) en dos partes iguales. Escoge la palabra de la tabla que queda en medio de las dos mitades y la explora. Unicamente pueden ocurrir tres casos:

1) Que la palabra explorada coincida con la de entrada (CLAVE).
2) Que CLAVE tenga un orden alfabético mayor que el de la palabra explorada.
3) Por último, que CLAVE tenga un orden alfabético menor.

En los dos últimos casos, una de las mitades de la tabla se rechaza. La búsqueda continúa dividiendo esa nueva mitad en dos partes iguales y se procede de la misma forma que antes hasta que se produzca el caso 1, es decir, que CLAVE coincida con la palabra que se explora.

Para la exploración de una mitad de la tabla se tienen dos *punteros,* uno que señala el principio de esa mitad (INICIAL) y otro el final (FINAL).

Antes de pasar a presentar el organigrama que corresponde a esa exploración, vamos a solucionar un caso que debe ser tenido en cuenta: cuando el número de elementos de la tabla es par, significa que no existe ninguna palabra que caiga justo en medio de la tabla. Lo que se hace en ese caso es redondear la división por defecto, o lo que es lo mismo, que el resultado de la división siempre sea un número entero.

El punto medio de la tabla se obtiene de la semisuma del puntero que indica el índice INICIAL y del que apunta a la última palabra.

(INICIAL + FINAL)/2 = Punto medio (siendo éste un entero).

El organigrama correspondiente se presenta en la figura 3.2. Donde:

INIC = Puntero que señala el principio de la parte de la tabla a explorar.
FINAL = Puntero que señala el final de la parte de tabla a explorar.
J = Indice de la palabra que se explora.
CLAVE = Palabra de entrada.
N = Total de palabras almacenadas.
L(J) = Palabra de la lista que ocupa la posición J.

Este algoritmo es más rápido en la mayoría de los casos, puesto que elimina en cada iteración la mitad de la tabla.

La primera decisión «¿el índice es igual al puntero INICIAL?» se efectúa para comprobar que la tabla que hay que explorar se ha convertido en una tabla sin longitud, es decir, que el puntero INICIAL y FINAL coinciden en un mismo lugar con lo cual termina la exploración.

Realmente, aunque el número de comparaciones sea menor en el algoritmo de exploración binaria que en el de exploración secuencial, las comparaciones que debe realizar la máquina son más fáciles en el caso del algoritmo secuencial que en el de exploración binaria.

### 3.2.3. Algoritmo binario con interpolación

El sistema que se emplea es el de realizar un pequeño cálculo para decidir cuánta separación alfabética existe sobre la palabra explorada y la CLAVE y por interpolación, acercarse más a la clave.

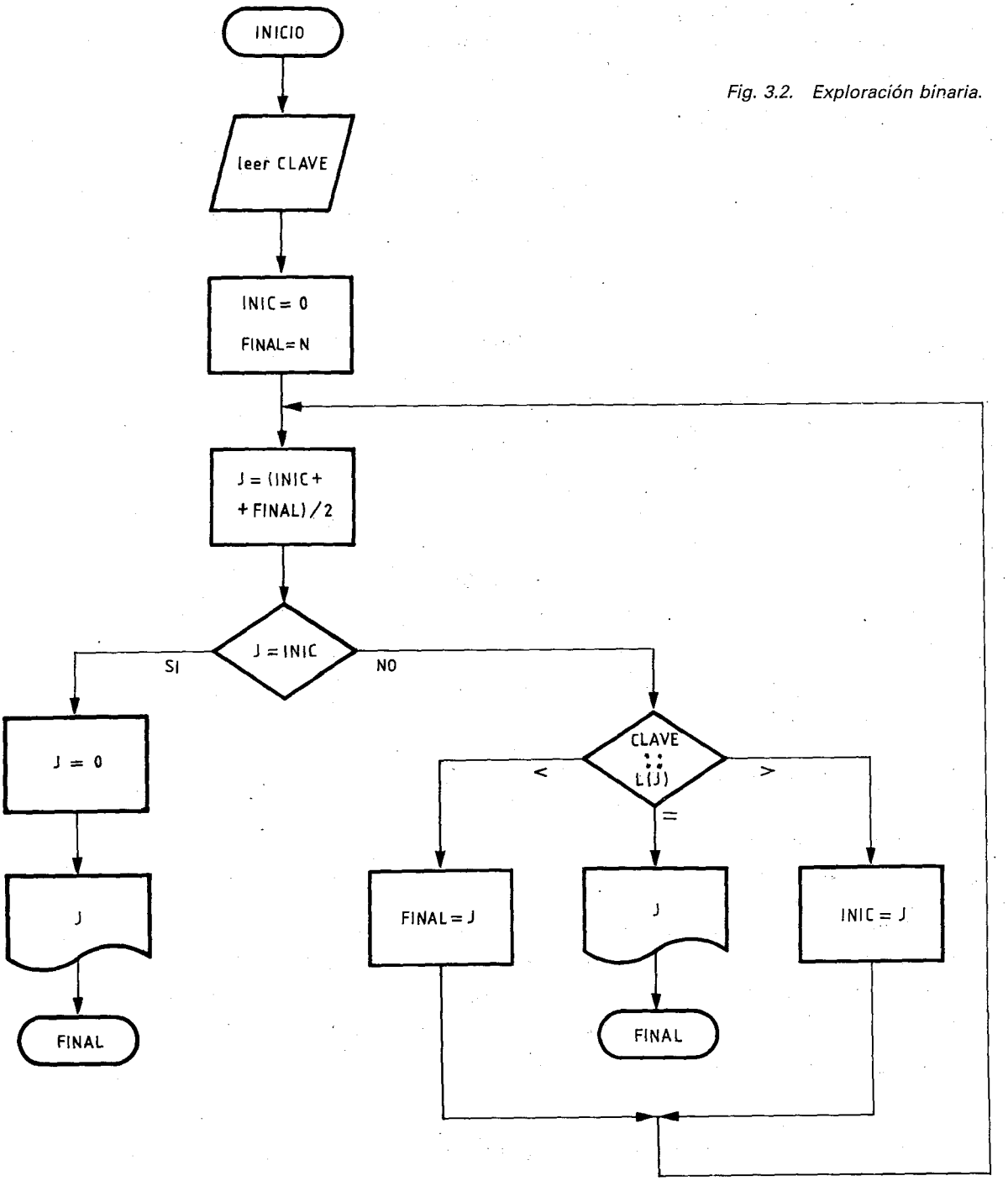



*Fig. 3.2. Exploración binaria.*

El cálculo da como resultado la colocación del puntero en el lugar que más se acerque a la palabra que se va a buscar. De este modo, si se empieza la exploración en la letra del alfabeto «S» y la clave pertenece a la letra «R», la interpolación decide que el puntero se mueva a la izquierda un número de palabras n/26. Siendo n el número total de palabras en la lista y 26 el número de letras del alfabeto.

En la práctica, este método no presenta demasiadas ventajas de ahorro de tiempo frente al binario, excepto en el caso de tablas muy extensas.

Este método resulta más lento que el de exploración binaria, ya que considera a las palabras distribuidas uniformemente en la tabla, lo que conduce a la máquina a suponer que hay tantas palabras que comienzan con la letra «S» como con la «R» o, en general, con cualquier letra del alfabeto. Esta condición no se cumple en la mayoría de los casos lo cual resta eficacia al método.

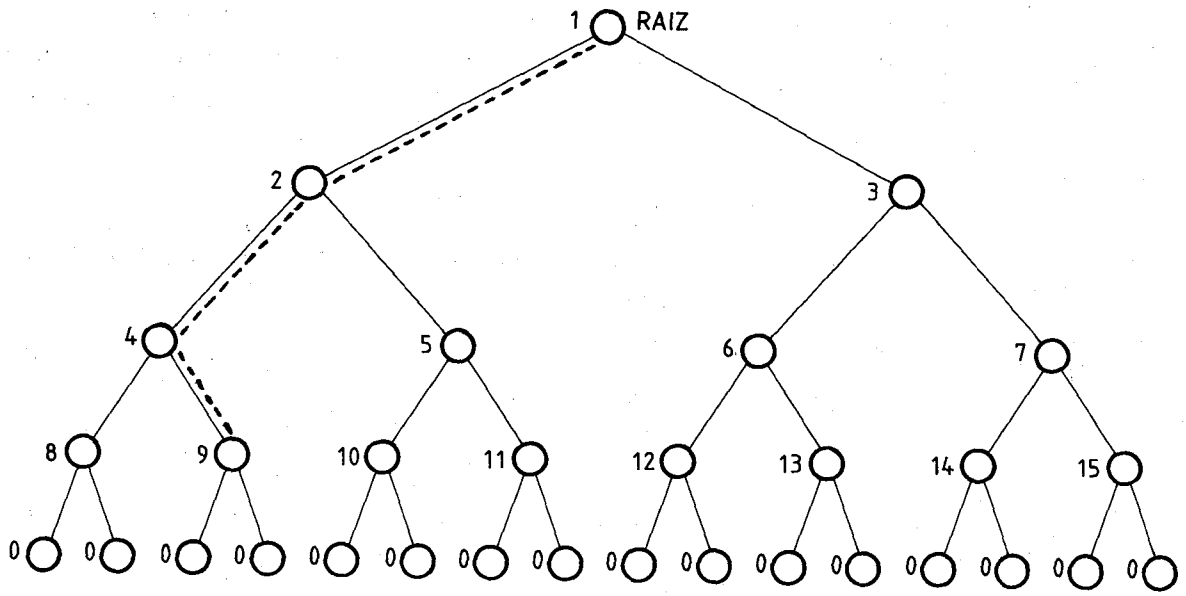



*Fig. Fig. 3.3. Arbol de exploración binaria en el que se ha marcado un camino (con línea discontinua) hasta el encuentro de la palabra con índice 9.*

### 3.2.4. Exploración en árbol binario

Los datos deben estar ordenados alfabéticamente. La eficiencia de este método depende de la ordenación que se dé a los datos.

Este algoritmo se llama *en árbol* porque la exploración de una palabra lleva directamente a la siguiente, siempre a la misma (como una rama que saliera de otra rama). Se llama binario porque de una palabra se puede tener acceso a *dos* en el paso siguiente, eligiéndose una u otra según sea el resultado de la comparación con CLAVE. En la figura 3.3 se ve que la estructura es en forma de árbol invertido.

Refiriéndonos a la figura, y empezando a explorar la palabra que ocupa la posición 1, ésta se compara con la CLAVE. Si la palabra explorada es posterior, en orden alfabético, a CLAVE, la siguiente palabra que se explora es la que se encuentra en la rama de la izquierda. Dicha palabra está marcada con un 2 en la figura. En caso contrario, es decir, en el caso de que CLAVE sea posterior, en orden alfabético a la palabra explorada, la rama que se toma es la de la derecha, llegando a la palabra marcada con un 3 que es la siguiente a explorar.

Para programar la estructura en árbol binario se necesitan dos punteros, uno que marca a la siguiente palabra por la rama izquierda (IZQUIERDA), y otro que marca a la siguiente palabra por la rama de la derecha (DERECHA). Una vez hecha la comparación con la clave se elige uno de los dos punteros según la regla siguiente:

Si CLAVE es posterior a la palabra explorada, entonces la siguiente palabra explorada es DERECHA.

Si CLAVE es anterior a la palabra explorada, entonces la siguiente palabra explorada es IZQUIERDA.

El organigrama de la exploración por árbol binario se presenta en la Figura 3.4.

En las últimas ramas del árbol (fig. 3.3) no existen palabras y quedan colocados todos los índices cero. La máquina acaba el proceso cuando encuentra la posición con el índice cero. Colocar palabras con índice cero (que en realidad no existen) permite añadir nuevos términos en la lista. Para ello se elimina uno de los ceros de las últimas ramas y en su lugar se coloca la nueva palabra.

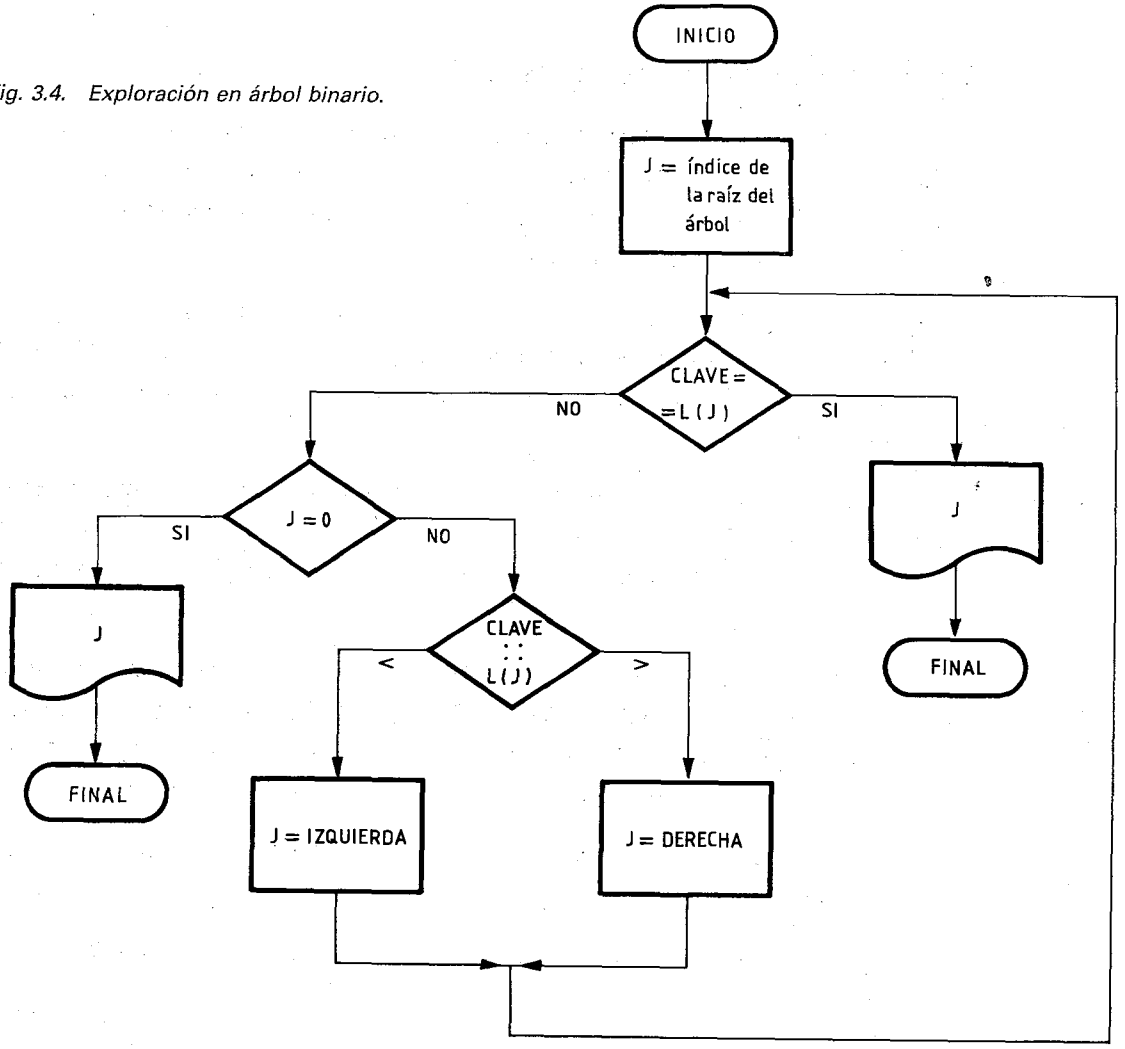



*Fig. 3.4. Exploración en árbol binario.*

Las mejoras que se pueden introducir en este algoritmo son debidas a distintas colocaciones de los elementos de la lista. Si se exploran primero las palabras de la lista que aparecen con más frecuencia llegaremos antes a encontrar la palabra buscada.

Para listas de palabras empleadas en el lenguaje corriente existen estudios de la probabilidad de aparición. Se elige una raíz que esté cerca de la mitad de

la tabla y cerca de la palabra con mayor probabilidad de aparición. A partir de esa raíz se iría desarrollando el árbol ordenándolo por probabilidades de aparición más altas.

### 3.2.5. Algoritmo de condensación

El método de *condensación* se basa en la organización de los datos en una tabla. El criterio de organización de la lista es asignar a cada palabra una posición según una función numérica, llamada *función de condensación,* que se le da a las letras de la palabra.

Una función de condensación consiste en dar un número a la palabra según su letra inicial. Así todas las palabras que empiezan por «a» tendrían la posición 1, todas las palabras que empiezan por «b», la posición 2 y así sucesivamente.

Por ejemplo, si se trata de encontrar la palabra CLAVE, en primer lugar se halla el valor de la función de condensación. Como CLAVE empieza por la letra C, su posición en la tabla estará indicada por un valor numérico que empiece por 3.

Las palabras cuya inicial sea la letra C estarán numeradas en orden ascendente (es decir, 30001, 30002, 30003, ...). El valor numérico que indica la posición de la primera palabra cuya inicial es C se sitúa en el campo MIN, y el valor que indica la última palabra de inicial C se sitúa en MAX.

El valor numérico que indica la posición de CLAVE debe encontrarse entre MIN y MAX.

El organigrama para dicha exploración se muestra en la figura 3.5.

Otra función de condensación más eficiente será aquella que tenga en cuenta todas las letras de la palabra. Por ejemplo, la posición de la palabra HACE estaría indicada con el valor numérico 8135 (correspondiendo los dígitos a cada una de las letras respectivamente).

## 3.3. CARACTERISTICAS DE UN PROGRAMA

En un programa deben tenerse en cuenta las siguientes características:

- La longitud o extensión del programa.
- El volumen de datos que se maneja.
- El tiempo de ejecución.

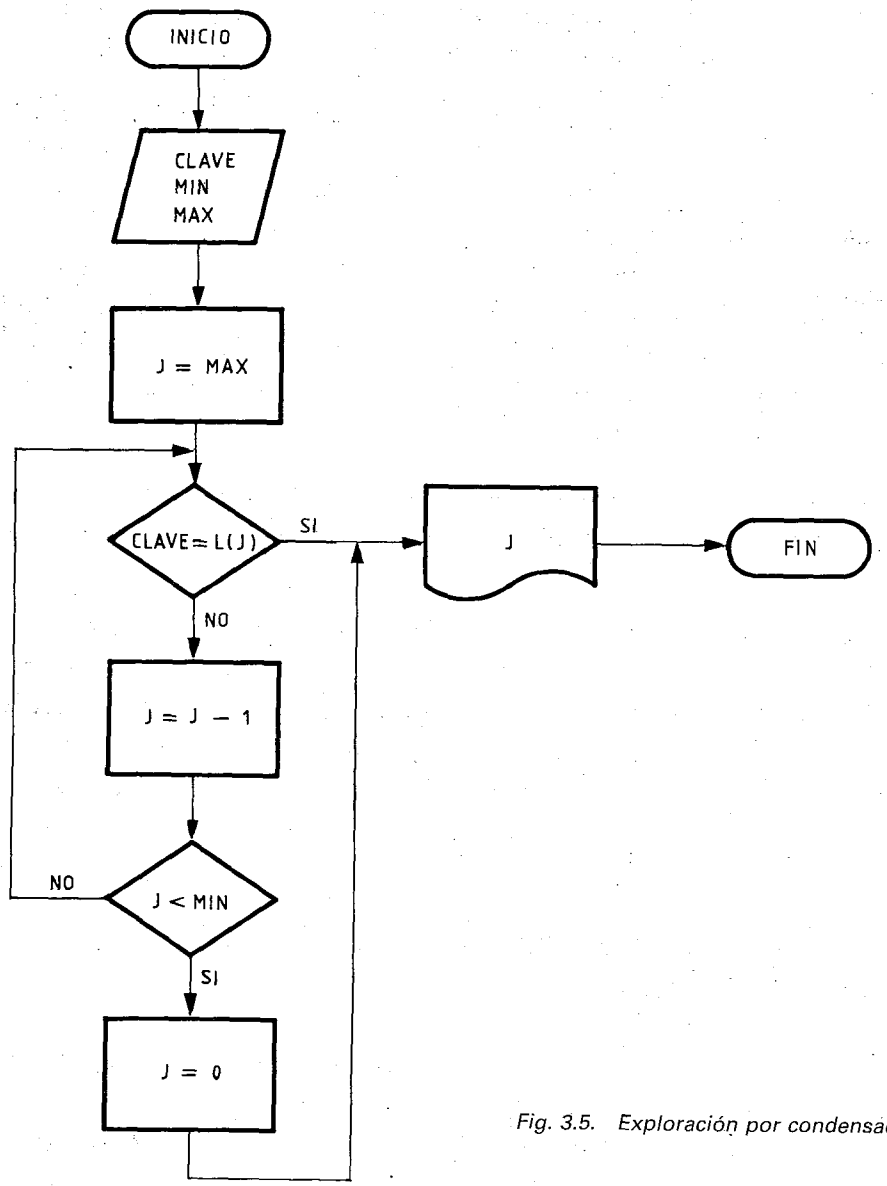



*Fig. 3.5. Exploración por condensación.*

Las características primera y segunda repercuten en la cantidad de memoria que va a usarse, teniendo en cuenta que los programas y los datos que se manejan se almacenan en la memoria para procesarse.

El tercer punto tiene interés en la consecución de un algoritmo que sea más eficiente por ser más rápido. Una deficiente orientación de las funciones que debe realizar la máquina puede dar lugar a una disminución importante del tiempo de respuesta.

La cantidad de memoria usada también debe ser minimizada por varias razones, una de ellas es por los costes: cuanta más memoria se utilice, más costará el equipo. El volumen de datos que debe manejar la máquina no puede modificarse, pero sí ir usándolo gradualmente para que no ocupe completamente la memoria. Normalmente, si este volumen de datos es grande, está almacenado en una memoria periférica de donde pueden irse obteniendo los datos a medida que los necesite.

Todas estas características dependen de la máquina, de lo que se deduce rápidamente que no todos los algoritmos que se haga podrán ser resolubles en una cierta computadora.

Lo dicho anteriormente supone una limitación a los programas que se pueden hacer. De todos modos, con una metodología lógica para la creación de programas pueden mejorarse las ocupaciones de memoria y el tiempo de ejecución. Antes de exponer esta metodología veamos cuáles son las fases por las que se debe pasar hasta la confección definitiva de un programa.

## 3.4. RITERIOS EN LA CONSTRUCCION DE PROGRAMAS

Se presentan en este tema las etapas que deben seguirse al confeccionar un programa y los criterios para que un programa sea eficaz. Asimismo se estudian las relaciones existentes entre los programas y las demás estructuras de información que existen en la máquina.

### 3.4.1. Criterio de normalización

Al construir el algoritmo no se tiene en cuenta el tipo de máquina que va a usarse, sin embargo, el programa ya está más cercano a la máquina y por lo tanto ha de conocerse y contar con ella para diseñarlo.

La idea actual de la programación es que sea lo más general posible, es decir, se intenta desarrollar programas que funcionen en todas las máquinas de tipo parecido. Desde luego un programa para una computadora es difícil que se adapte sin más a un microcomputador pero dentro de los ordenadores de potencia parecida debe ser compatible. Esta compatibilidad se dio sobre todo, al crear los lenguajes de alto nivel. En esos lenguajes las instrucciones están encaminadas a resolver procedimientos más que a la transferencia de información de un registro a otro como ocurre en los lenguajes orientados a la máquina. En lenguajes de alto nivel se resuelven situaciones como por ejemplo, escribir el resultado de un determinado cálculo, mientras que en lenguajes orientados a la máquina sería «mover el contenido del registro *x* al registro de dirección *d*».

La normalización aún no se ha conseguido y se está intentando crear procedimientos para que, encadenados de forma adecuada resuelvan todas las situaciones posibles. Esto repercutiría en una facilidad en la confección del organigrama, ya que éste debe responder a esas estructuras o procedimientos y en una más fácil codificación, porque se tendrían instrucciones o grupos de instrucciones que corresponderían exactamente a esa estructura, las cuales sólo tendrían que colocarse en el lugar adecuado.

### 3.4.2. Criterio de realización

En la realización de un programa se han de prever las acciones oportunas para toda eventualidad que pudiera presentarse.

Es decir, estudiar todas las distintas posibilidades que pudieran presentarse y tenerlas en cuenta a la hora de confeccionar el programa.

Veamos esto con un ejemplo. Supongamos que tenemos que resolver una ecuación de segundo grado.

La forma de una ecuación de segundo grado es:

$$ax^2 + bx + c = 0$$

donde *a, b, c* son parámetros (constantes) y la variable es *x*.

La ecuación tiene dos soluciones que son

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

El organigrama será el que presenta la figura 3.6, según este organigrama la máquina se parará para algún juego de parámetros, como por ejemplo, *a, b, c* todos iguales a 1, ya que no puede resolver la raíz cuadrada de números negativos (en este caso de −3). Puesto que no existe una raíz real de un número negativo, las soluciones son números complejos, no reales, que la máquina no posee interiormente.

Para completar el ejemplo las soluciones serán en este caso:

$$x_1 = \frac{-\,b}{2a} + \frac{\sqrt{(b^2 - 4ac)}}{2a}$$

$$x_2 = \frac{-\,b}{2a} - \frac{\sqrt{(b^2 - 4ac}}{2a}$$

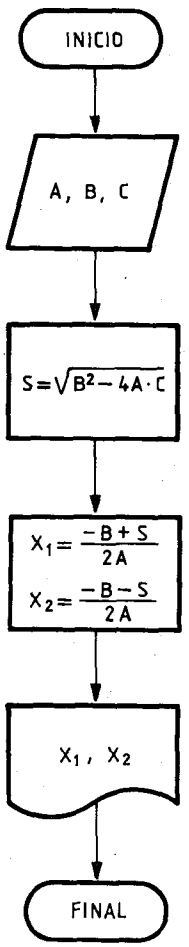



*Fig. 3.6. Resolución parcial de la ecuación de 2.° grado.*

Eso significa que para evitar esa eventualidad en el programa se debe prever que aparezcan radicandos negativos poniendo, por ejemplo, una condición que permita continuar el programa o pararlo diciendo que las soluciones no son reales (muy fácilmente podrían calcularse las soluciones complejas).

Estudiando un poco más profundamente el ejemplo puede verse que la máquina da resultados erróneos cuando se le presentan ternas *a, b,* ∅. Esto se debe a que a la ecuación de segundo grado le falta el término *c* y el modo de cálculo general presentado no es válido. Esto sería motivo de error y no se había tenido en cuenta al confeccionar el algoritmo.

Por otra parte, cuando $a = \emptyset$ la solución tampoco puede obtenerse con la fórmula presentada puesto que la ecuación ya no es de segundo grado sino que es de primer grado.

El organigrama completo contando con todas las irregularidades se muestra en la figura 3.7.

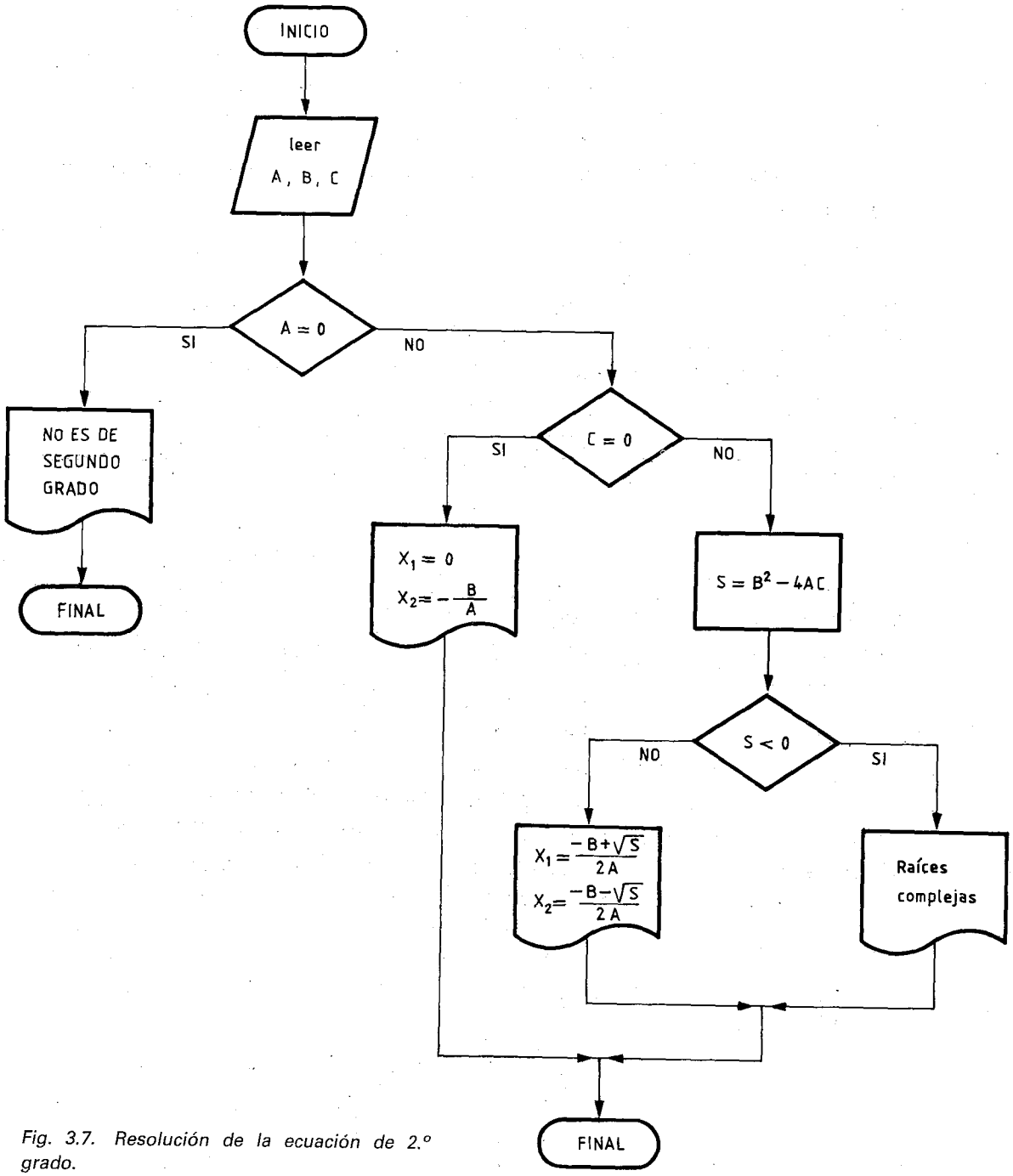



*Fig. 3.7. Resolución de la ecuación de 2.º grado.*

El programa del ejemplo anterior, sin ser ampliado puede dar el resultado correcto mientras no aparezca en la entrada una combinación peculiar de valores. Eso es lo que se llama corrección según el dominio de valores de entrada.

Cuando un programa se documenta, es decir, al añadirle comentarios, hay que hacer mención al dominio de valores sobre los que puede trabajar sin error.

En la documentación del programa del cálculo de una ecuación de segundo grado debe ponerse:

```
11  REM PROGRAMA PARA EL CALCULO DE LAS SOLUCIONES DE
12  REM LA ECUACION DE SEGUNDO GRADO.
13  REM LOS COEFICIENTES DE LA ECUACION SON A, B, C.
14  REM LAS SOLUCIONES SON X1 y X2.
15  REM EL JUEGO DE VALORES DE ENTRADA NO DEBE POSEER.
16  REM A = Ø ni C = Ø.
```

Aquí no se ha incluido el caso de una terna de valores de entrada que den soluciones no reales. Esas ternas de valores sí que serían posibles y el programa es quien se encargaría de identificarlas y dar un comentario de salida adecuado.

Con esos comentarios el programa queda de la forma primitiva y únicamente se le añade una ampliación para que verifique el signo del radicando. En función de ese signo continuaría o no el proceso, siendo el signo positivo o negativo respectivamente. Si el proceso se para (signo negativo) se da un mensaje de salida explicando las razones de no concluir el proceso.

## EJERCICIOS

1. El interés que tiene un algoritmo es,

   *a)* Que alguna vez se acaba, al encontrar la solución.
   *b)* Que siempre está calculando.

2. Un algoritmo es exclusivo del manejo por ordenador

   *a)* No, puede desarrollarse a mano.
   *b)* Sí, porque el ordenador es el único que puede manejar algoritmos.

3. Cuando diseñamos un algoritmo.

   *a)* Basta con que obtenga los resultados.
   *b)* Puede mejorarse si se estudia más profundamente.

4. Para los problemas que manejan datos:

   *a)* Un detalle muy importante es la organización de los datos.
   *b)* Los datos pueden presentarse de cualquier manera, el ordenador está para tratarlos estén como estén.

5. En el algoritmo secuencial el número de comprobaciones que se hacen es:

   *a)* Una, como máximo.
   *b)* La mitad de las palabras de la tabla, como máximo.
   *c)* La totalidad de las palabras de la tabla, como máximo.

6. En el organigrama para el algoritmo secuencial, ¿cómo averiguamos que ya no tenemos que hacer más comprobaciones?

   *a*) Cuando J = ∅.
   *b*) Cuando CLAVE = L(J).
   *c*) Ambas a la vez.

7. En el algoritmo de exploración binaria el punto siguiente de exploración se calcula:

   *a*) Del valor del puntero final.
   *b*) De la semisuma de los valores de los dos punteros.

8. ¿Cómo obtenemos los nuevos punteros?

   *a*) Verificando en qué mitad de la tabla se encuentra el resultado.
   *b*) Incrementándolos sucesivamente.

9. ¿Cuándo se acaba de explorar?

   *a*) Cuando los dos punteros coincidan en el mismo lugar.
   *b*) Cuando hemos encontrado el lugar que ocupa la palabra CLAVE.
   *c*) Ambas a) y b).

10. La ventaja mayor de la exploración binaria con interpolación se obtiene

    *a*) Cuando las palabras crecen alfabéticamente.
    *b*) Cuando las palabras están distribuidas uniformemente en todo el orden alfabético.

11. La exploración en árbol binario:

    *a*) Es la más rápida.
    *b*) Lleva mucho tiempo de cálculo del siguiente valor.

12. La raíz del árbol es la palabra

    *a*) Que se encuentra en mitad del árbol.
    *b*) Que tiene más probabilidades de producirse.
    *c*) Que tiene el orden alfabético cercano a la mitad y más probabilidades de producirse.

13. El algoritmo de condensación

    *a*) Estudia primero la clave antes de ir a buscarla.
    *b*) Va a buscar una palabra directamente en la lista.

14. Debemos pensar en la normalización como tendencia deseada de todos los programadores.

    *a*) Sí, porque no estarán tan limitados por la máquina a usar.
    *b*) No, puesto que las máquinas siempre van a ser diferentes.

15. Un programa será más eficiente si

    *a*) Tiene menor longitud realizando los mismos cálculos.
    *b*) Tarda menos en realizar las mismas tareas que otro.
    *c*) Ambas condiciones.

16. Un programa que tenga en cuenta todos los casos posibles que puedan presentarse

    *a*) Es bueno pero habría que ver de qué modo resuelve esos casos.
    *b*) Es siempre bueno.

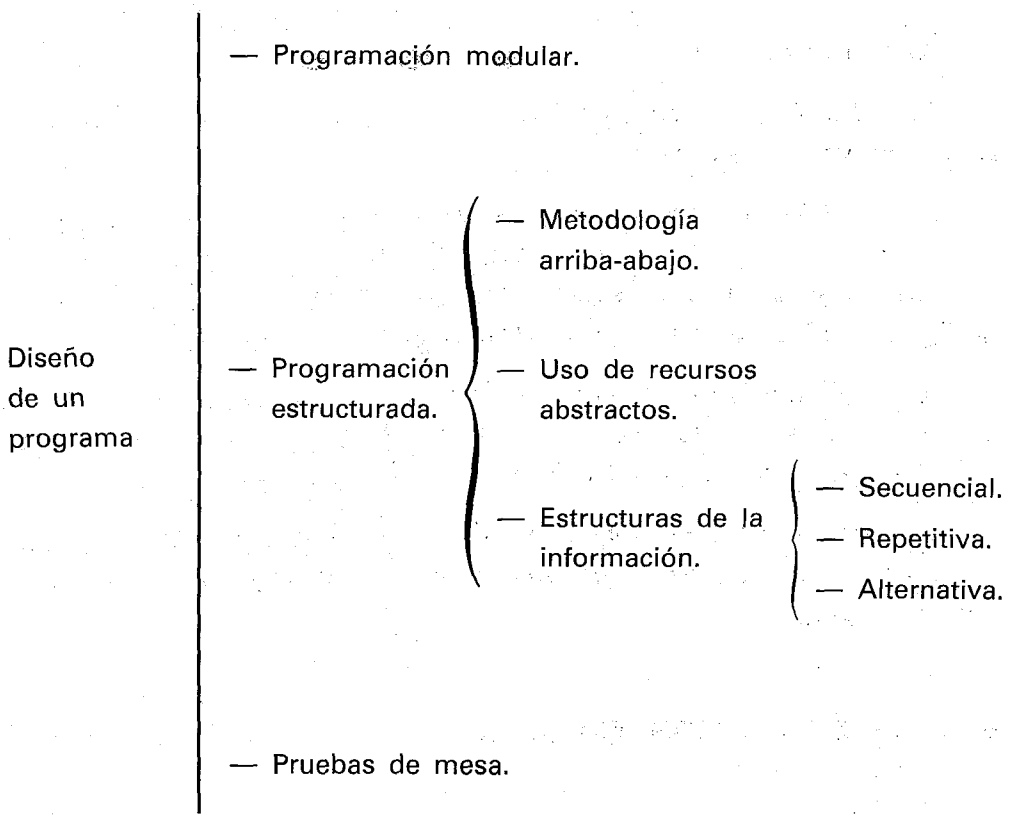
17. ¿Un problema puede considerarse correcto si se cumplen todos los casos posibles?

    *a*) Sí, si hemos definido el rango de valores de entrada.
    *b*) No, porque debe tomar en cuenta todos los casos.

# 4.

# DISEÑO DE UN PROGRAMA

## CUADRO RESUMEN DE LA LECCION



Después de haber conocido cuáles son las características que debe tener un programa se puede empezar a describir una metodología encaminada a conseguir esas premisas.

Cualquier programa debe ocupar la mínima memoria posible y ejecutarse con el mínimo tiempo de máquina.

La metodología que va a emplearse es la de la *programación lógica.* Las técnicas de este método se conocen como *programación modular, programación lógica* o *programación estructurada.* La diversidad de nombres se debe a un ligero cambio de matiz que se desarrollará a continuación.

Se puede considerar la programación lógica como todo el conjunto de técnicas para la creación de un programa eficiente. Esas técnicas serían aplicables desde la fase de diseño hasta la fase de documentación.

La Programación lógica puede consistir, por ejemplo, en: usar la programación estructurada, documentar un programa, diseñar un modo eficiente de probar un programa.

## 4.1. PROGRAMACION MODULAR

La programación modular incidiría sobre todo en la codificación y en el diseño.

En el diseño porque debería descomponerse el problema en módulos o tareas más simples. Cada tarea representa una actividad completa y se codifica separadamente. El programa principal tiene como única misión coordinar la

adecuada llamada a las subrutinas con las que se han codificado las pequeñas tareas.

Este modo de estructuración permite que la puesta a punto del programa sea mucho más cómoda y fácil porque se ponen a punto cada uno de los trozos por separado. Una vez funcionando cada una de las subrutinas podemos ensamblarlas y darle el control de todas ellas al programa principal.

Una técnica muy conveniente e idónea para la producción de los programas planteados de esta manera es la de segmentación. Utilizando el montaje por segmentación podemos ahorrar espacio en la memoria, lo que nos será muy conveniente en multitud de aplicaciones.

Veamos un ejemplo de programación modular.

Supongamos una determinada frase y que deben escribirse cada una de sus palabras en columna y ordenadas alfabéticamente.

Los pasos para la resolución de este problema serán los siguientes:

1) Lectura de la frase.
2) Separar las palabras de la frase.
3) Ordenarlas alfabéticamente.
4) Escribirlas en columna.

Con estos cuatro puntos el problema está resuelto. El siguiente paso consiste en desmenuzar cada uno de estos puntos.

La ventaja que se tiene con la utilización del método es que cada uno de los apartados en que se ha dividido el problema constituyen un problema aparte.

Si la frase de entrada es:

JUAN ESTUDIA MATEMATICAS

La salida debe ser:

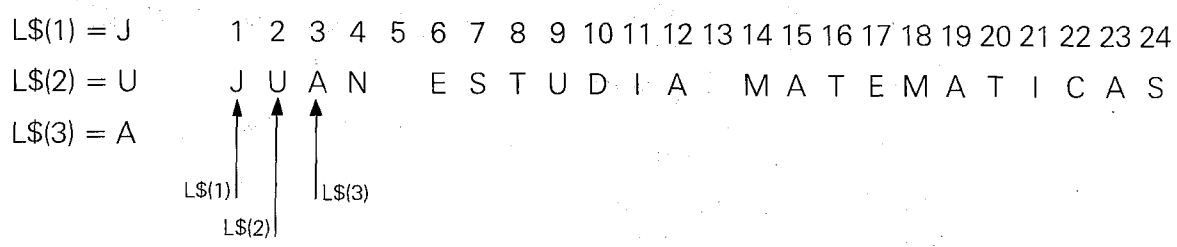
ESTUDIA
JUAN
MATEMATICAS

### *PASO 1: Lectura de la frase*

La lectura se hace directamente asignando cada una de las letras de la frase a un vector alfanumérico (por ejemplo, L$).

El único esfuerzo de diseño que se ha tenido que hacer es decidir que va a ser un vector alfanumérico quien va a recoger la frase, letra por letra. El motivo de elegir esta estructura de datos, es que en pasos sucesivos debe analizarse cada una de las letras para poder separar palabras. La variable debe ser alfanumérica y además se utiliza un vector para que nos sea fácil pasar a la letra siguiente.

La frase que se ha tomado como ejemplo quedaría almacenada de la siguiente manera:




### *PASO 2: Separación de las palabras de la frase.*

El procedimiento para la realización de la separación es explorar la lista de letras hasta encontrar un espacio en blanco. Una vez encontrado, la palabra vendrá delimitada por este espacio en blanco. En este caso: L$(5) = " ". Por lo tanto, la palabra irá desde L$(1) hasta L$(4).

Se copia esta palabra en una matriz de dos dimensiones en la que en cada fila habrá una palabra de la frase.

En el ejemplo la matriz quedará escrita como sigue: (llamando a la matriz N$(I,J)).

| N$(I,J) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | J | U | A | N | | | | | | | |
| 2 | E | S | T | U | D | I | A | | | | |
| 3 | M | A | T | E | M | A | T | I | C | A | S |

### *PASO 3: Ordenación alfabética*

Puesto que las frases nunca van a ser muy largas, el algoritmo que se encarga de la ordenación será simplemente uno de búsqueda de la palabra que ocupa el primer lugar en el orden alfabético, colocando a ésta en la primera posición de la matriz.

A continuación se hace lo mismo con las restantes palabras y, de este modo, se van colocando cada una de ellas en su nueva posición, hasta que estén totalmente ordenadas.

### *PASO 4: Escritura en columna*

Este paso consiste en organizar un programa que liste una matriz. Ese listado nos dará las palabras en columna y ordenadas alfabéticamente.

El listado será el siguiente:

```
E S T U D I A
J U A N
M A T E M A T I C A S
```

## 4.2. PROGRAMACION ESTRUCTURADA

Cuando alguien se pone a programar por primera vez, lo normal es que vaya desarrollando el problema según una secuencia natural en el tiempo. Es decir, que se ponga en el primer punto e intente regenerar todos los pasos que va a realizar la máquina.

Si se emplea esta *programación natural* (se utilizará esta palabra siempre que se refiera en el texto al método de simular que el programador es el ordenador y seguir un proceso secuencial) se invierte el mismo tiempo en programar los detalles que en programar las partes fundamentales.

En la programación natural a medida que van apareciendo las tareas se van resolviendo sin distinguir si es una tarea fundamental o un simple detalle. En la programación estructurada se hace *a priori* una organización de las tareas, descomponiendo dichas tareas principales y generales en otras cada vez más simples.

En la programación estructurada es conveniente que todos los programadores usen unas mismas estructuras de programación. Unas estructuras que resuman todas las posibles situaciones de programación que puedan presentarse. De este modo se deduce que la estructuración es un paso adelante en la unificación de todos los programadores, con las ventajas que esto representa para el intercambio de información y programas.

La programación estructurada permite comparar dos programas si en ambos se han usado las mismas estructuras.

El procedimiento empleado en la programación estructurada, además de usar en la construcción de los organigramas una serie de estructuras básicas, es el de dar varios pasos hasta encontrar la solución. De un paso se obtiene el siguiente dando un mínimo de información sobre lo que debe realizar el programa, y así sucesivamente hasta que el problema está acabado completamente.

Las normas a seguir en la confección de los algoritmos dentro de la programación estructurada son las siguientes:

— Los pasos se desarrollan en descripciones sucesivas empezando con la descripción Ø.

— En cada uno de los pasos se añade más información sobre la resolución del problema.

— En la descripción Ø sólo se pone el enunciado del problema.

— Las palabras EMPIEZA y FINAL señalan el inicio y fin del algoritmo.

*Ejemplo*

Escribir una tabla de los 1ØØØ primeros números primos (siendo 2 el primero de ellos).

Descripción Ø:

EMPIEZA
    Escribir los 1.000 primeros n.$^{os}$ primos empezando por el 2.
FINAL

Descripción 1:

EMPIEZA
1. Variable: tabla de números primos.
2. Rellenar la tabla con los 1ØØØ primeros n.$^{os}$ primos.
3. Escribir la tabla.

FINAL

En esta descripción la única unión entre las dos tareas es la tabla de los números primos. Si se consigue definir cómo es la tabla se podrán separar en dos tareas independientes.

El punto 2 se desarrolla con la descripción 2:

Descripción 2

EMPIEZA
1. Conjunto de 1ØØØ primeros números primos.
2. *Hacer:*
   K = 1 hasta K = 1ØØØ
   P(K) = número primo que ocupa el lugar K.
3. *Escribir:* P(K) desde K = 1 hasta K = 1ØØØ.

FINAL

Se puede seguir desmenuzando el problema, ahora separando completamente el punto 2 y el 3.

Descripción 3 (punto 2)

EMPIEZA
P(1) = 2, P(2) = 3, P(3) = 5,...
FINAL

Descripción 4 (punto 2)

EMPIEZA
1. Enteros K, J; K = Ø, J = 1.
2. *Mientras* K sea menor que 1ØØØ *hacer:*
   2.1 Aumentar J hasta el siguiente número primo.
   2.2 K = K+1, P(K) = J

FINAL

Veamos ahora cómo incrementar J:

Descripción 5 (punto 2.1)

EMPIEZA
1. Variable: J primo
2. *Repetir*:
   2.1 J = J+1.
   2.2 Dar a J el significado:
      2.2.1 J es un número primo
3. *Hasta*:
   3.1 J sea un número primo

FINAL

Continuando con este procedimiento se resuelve el problema. Se resuelven todas las tareas por separado y luego se unen para formar el programa completo.

Así pues, la programación es estructurada si cumple los siguientes requisitos:

*a*) Se basa en un diseño de *arriba hacia abajo.*

*b*) Hace uso de recursos abstractos.

*c*) Hace uso únicamente de las tres estructuras básicas.

— Secuencial.
— Repetitiva.
— Alternativa.

### 4.2.1. Metodología arriba-abajo

La metodología de diseño *arriba-abajo* (o «TOP-DOWN», en inglés) consiste en efectuar una relación entre las etapas de la estructuración, de forma que una etapa jerárquica y su inmediata inferior se relacionen mediante entradas y salidas de información. Esto es, se considera al proceso jerárquico superior como suministrador de las entradas con las que se ejecuta el proceso jerárquico inmediatamente inferior, y una vez ejecutado se le devuelven al proceso jerárquico superior las salidas, que es el resultado del proceso.

### 4.2.2. Uso de recursos abstractos

El proceso comentado anteriormente de realizar sucesivos pases hasta encontrar la solución del problema, es un proceso abstracto. Este es el tercer punto que debe cumplir la estructuración: el uso de recursos abstractos.

Cualquier máquina podría reproducir los pasos, porque están especificados de modo abstracto. El proceso está resuelto en cada uno de los pasos que se dan. En estos pasos se sabe lo qué se tiene que hacer, pero no se conoce el detalle de cómo se va a hacer.

Supongamos el ejemplo de los números primos y qué elegimos la descripción 1. El programa contiene una subrutina llamada RELLENA y que se encarga de calcular los números primos.

Esta subrutina se desarrolla en la descripción 2 y en las sucesivas. Una vez la subrutina ha calculado los números primos devuelve el control al programa principal que la llamó y se pasa a la tarea siguiente: escribir una tabla, que a su vez también puede ser encomendada a una subrutina.

Avanzando más en las descripciones nos daremos cuenta de que también cada una de las subrutinas se pueden desmenuzar en otras.

Una vez vista la metodología que se va a emplear, se describen cuáles son las estructuras básicas de la información.

### 4.2.3. Estructuras de la información

Un programa realizado en programación estructurada debe tener únicamente tres tipos de estructuras en su diseño:

— Secuencial.
— Repetitiva.
— Alternativa.

#### *Estructura secuencial*

Una estructura secuencial es aquella que se sigue paso a paso sin bifurcaciones. Las tareas se suceden unas a otras siendo la salida de una, la entrada de la siguiente y así sucesivamente hasta el fin de la tarea. Todas estas estructuras tendrán una entrada y una salida.

Un ejemplo de un programa que sólo tuviera estructura secuencial sería el siguiente: "Calcular la suma de dos números cualquiera».

La descomposición de este problema según la programación estructurada sería la que se muestra a continuación:

Descripción 1

EMPIEZA
Sumar 2 números
FINAL

Descripción 2

EMPIEZA
Leer los números
Efectuar la suma
Escribir lá suma
FINAL

Descripción 3

EMPIEZA
Leer un número A
Leer el otro número B
Sumar C = A+B
Escribir el resultado C
FINAL

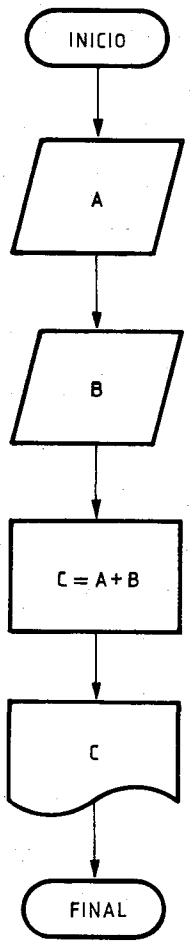



*Fig. 4.1.*

El organigrama correspondiente muestra la secuencia que en ningún momento se rompe, (Fig. 4.1).

## *Estructura repetitiva*

Esta estructura consta de un tratamiento o secuencia de tratamientos que se realiza de forma iterativa un número finito de veces; mientras se cumpla, o hasta que se cumpla una cierta condición.

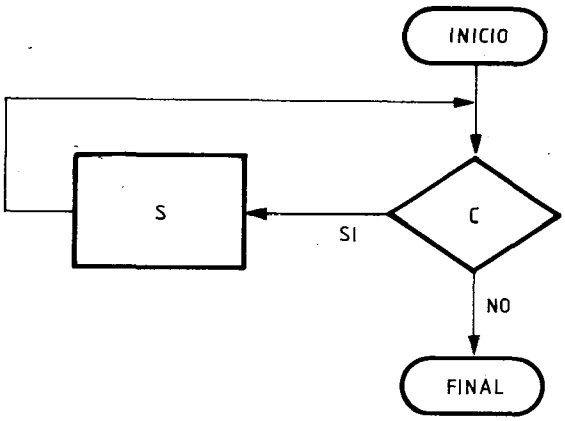
En este caso en la estructura existe un *tratamiento,* o una serie de tratamientos que deben hacerse y además una *condición* que debe cumplirse, de modo que en esta estructura ya existe una ruptura de secuencia que viene dada por la condición.

La estructura puede presentar diversas variantes que se muestran a continuación:

*a*) *Hacer* la tarea S *mientras* se cumpla la condición C. (Fig. 4.2).

*b*) *Repetir* la tarea S *hasta* que se cumpla la condición C. (Fig. 4.3).

*c*) *Hacer* la tarea S *mientras no* se cumpla la condición C. (Fig. 4.4).




◀ *Fig. 4.2.*

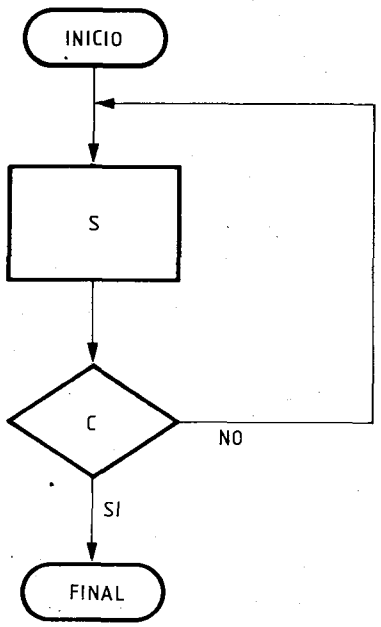
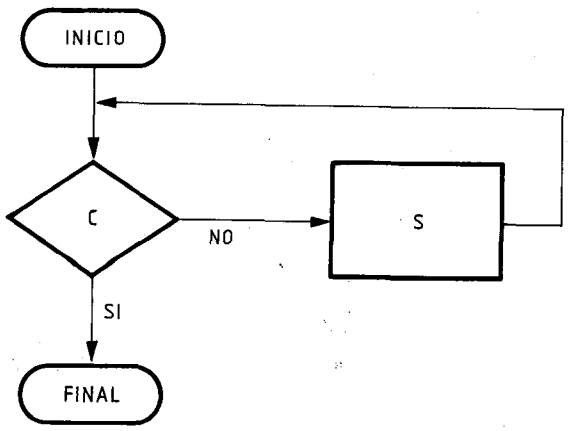



*Fig. 4.3.* ▶




◀ *Fig. 4.4.*

En las salidas de la condición pueden darse dos situaciones:

...*hasta* que se cumpla una condición
...*mientras* no se cumpla una condición.

En el caso *a*) *Hacer* la tarea S, *mientras* se cumpla la condición C, el rango de ejecuciones del tratamiento es de cero a N veces (considerando que se ha partido de cero), Esto es, que la tarea se realiza N+1 veces.

En el caso *b*) *Repetir* la tarea S *hasta* que se cumpla la condición C, el rango de ejecuciones es de 1 a N veces.

En el caso *c*) *Hacer* S *mientras no* se cumpla la condición C, el rango de ejecuciones es de Ø a N veces. Esta estructura equivale exactamente a la de «*mientras* se cumpla una condición *hacer* la tarea S». Por lo tanto, cualquiera de estas estructuras es intercambiable.

Veamos un ejemplo donde se usan estas estructuras (los más adecuados son los de tratamientos de listas):

*Ejemplo* 1

Dada una lista de 12 caracteres, averiguar en qué posición se encuentra una letra J.

Para no introducir nuevas estructuras en el ejemplo debe ponerse como condición del problema que el rango de entrada sea un conjunto de caracteres con, al menos, una «J».

A continuación se presenta el desarrollo del problema y en la figura 4.5 el organigrama correspondiente:

Descripción Ø:

EMPIEZA
    Buscar la posición de una J.
    Escribir dicha posición.
FINAL

Descripción 1:

EMPIEZA
*Repetir:*
Leer el siguiente carácter.
*Hasta:*
Caracter = J
Escribir posición de ese carácter.
FINAL

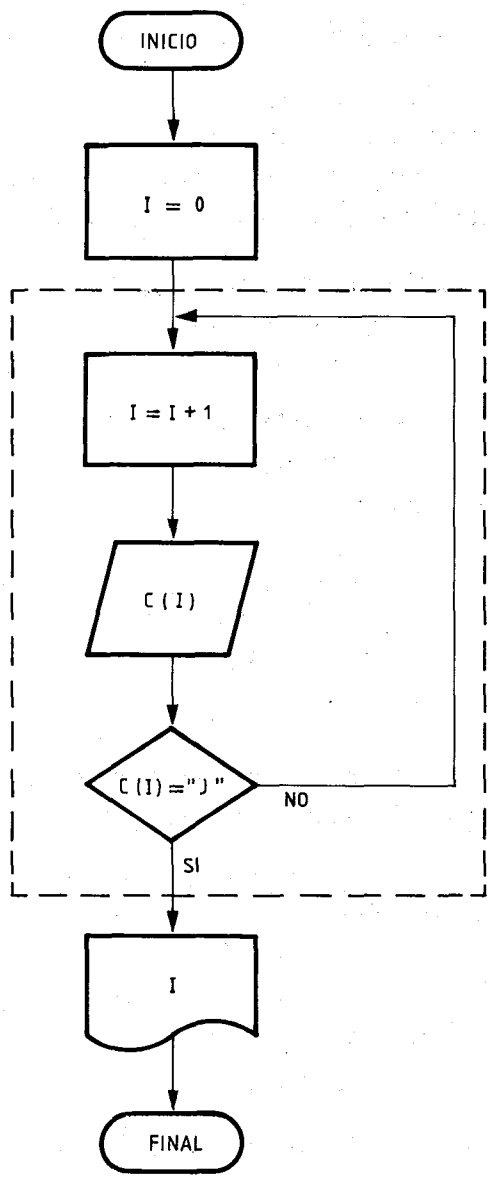



*Fig. 4.5. Encontrar la posición un carácter J en una lista de 12 caracteres.*

## *Estructura alternativa*

Por último, la estructura alternativa, contempla la actuación de dos o más tratamientos que, en la ejecución, se excluyen en función de una condición. Se debe ejecutar en cada caso un sólo tratamiento en función de la condición.

En el caso de dos tratamientos, caso más general, un tratamiento se ejecutaría cuando *se cumpliera* la condición y el otro se ejecutaría cuando *no se cumpliera* dicha condición.

Las distintas configuraciones que pueden presentarse se explican a continuación:

*Caso A*

*Si* se cumple la condición C *entonces hacer* el tratamiento A, *en caso contrario hacer* el tratamiento B.

En este caso se verifica el resultado del test (o condición): Si el resultado del test es afirmativo (la condición es verdadera) se ejecuta el tratamiento A.

Si el resultado del test es negativo (la condición es falsa, o no se cumple) el tratamiento que se ejecuta es el B. (Fig. 4.6).

*Caso B*

Este es un caso particular del anterior. En él una de las ramas de la alternativa está vacía.

*Si* se cumple la condición C *entonces hacer* el tratamiento S.

La ejecución del tratamiento sólo se hace en el caso de que la condición se cumpla, en caso negativo el programa sigue el flujo normal. (Fig. 4.7).

*Caso C*

La multitarea se presenta bajo la forma de una condición que puede tomar N valores enteros distintos: 1,2,3,4, ..., N. Según se elija uno de estos valores en la condición, se realizará una de las N tareas (recordemos que sólo se ejecuta *una* de las tareas *cada vez*).

La estructura puede escribirse como: *Hacer* ($S_1$, $S_2$, $S_3$, ..., $S_N$) *según* I. Donde I es un entero que puede tomar los valores 1,2,3,...,N. El esquema de esta estructura se representa en la figura 4.8.

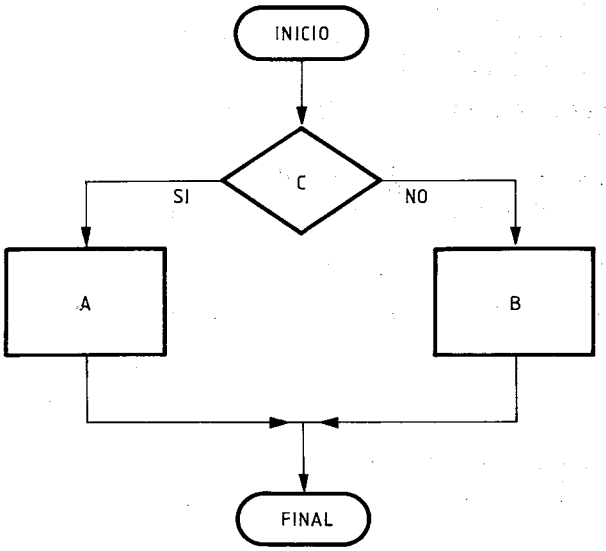



*Fig. 4.6. Estructura alternativa.*

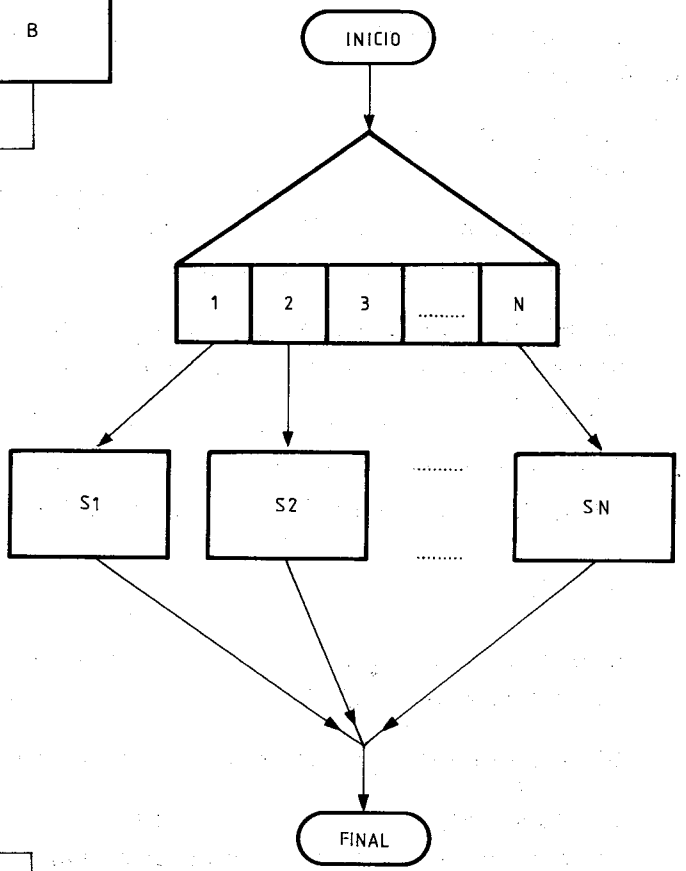



*Fig. 4.8. Estructura alternativa multitarea.*

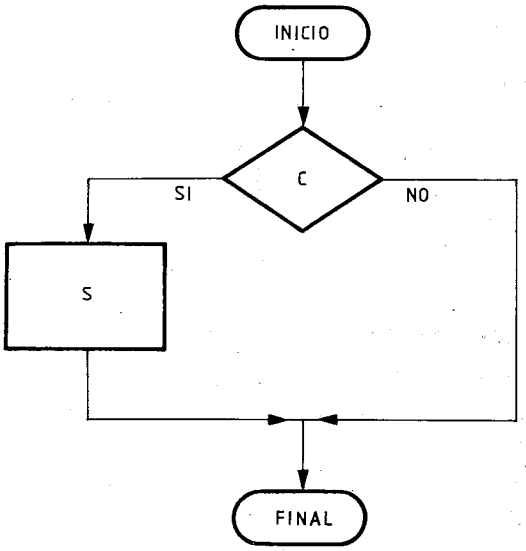



*Fig. 4.7. Estructura alternativa con una rama vacía.*

La diferencia entre las estructuras repetitivas y las alternativas es que las *repetitivas* ejecutan *varias veces* el tratamiento que tienen encomendado (lo ejecutan tantas veces como indique la condición) mientras que las *alternativas* sólo se ejecutan *una vez*.

En el caso de las alternativas, la condición viene determinada por un paso anterior en el programa, eligiéndose, según la condición, una de las tareas a ejecutar.

*Ejemplo* 1

Buscar el mayor de dos números, A y B.

Las descripciones para el desarrollo del problema son las siguientes:

Descripción Ø:

EMPIEZA
    Cálculo del mayor de dos números, A y B.
FINAL

Descripción 1:

EMPIEZA
    Si (A > B) hacer:
        escribir A.
    En caso contrario:
        escribir B.
FINAL

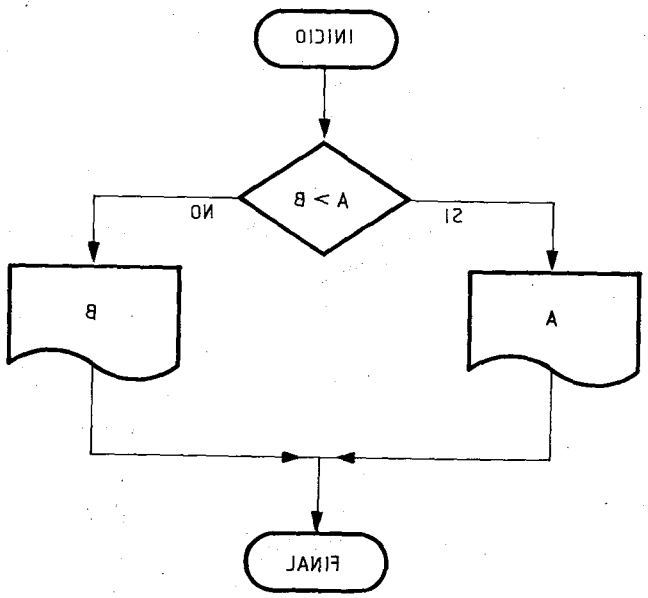



*Fig. Fig. 4.9. Cálculo del mayor de dos números.*

*Ejemplo* 2

Hacer varios listados de una lista única de números según el siguiente criterio:

— Si los números están comprendidos entre cero y cinco, ponerlos en la lista 1.
— Si están entre cinco y diez, en la lista 2.
— Si están entre diez y quince, colocarlos en la lista 3.

Nota: El cero pertenece a la lista 1, el cinco a la lista 2, el 10 y el 15 a la lista 3. La lista de entrada constará de 1Ø números.

Este problema combina las dos expresiones de estructura que nos quedan, la de la forma alternativa con una rama vacía y la multialternativa. No obstante, se verá que estas estructuras pueden combinarse siguiendo el método de programación estructurada.

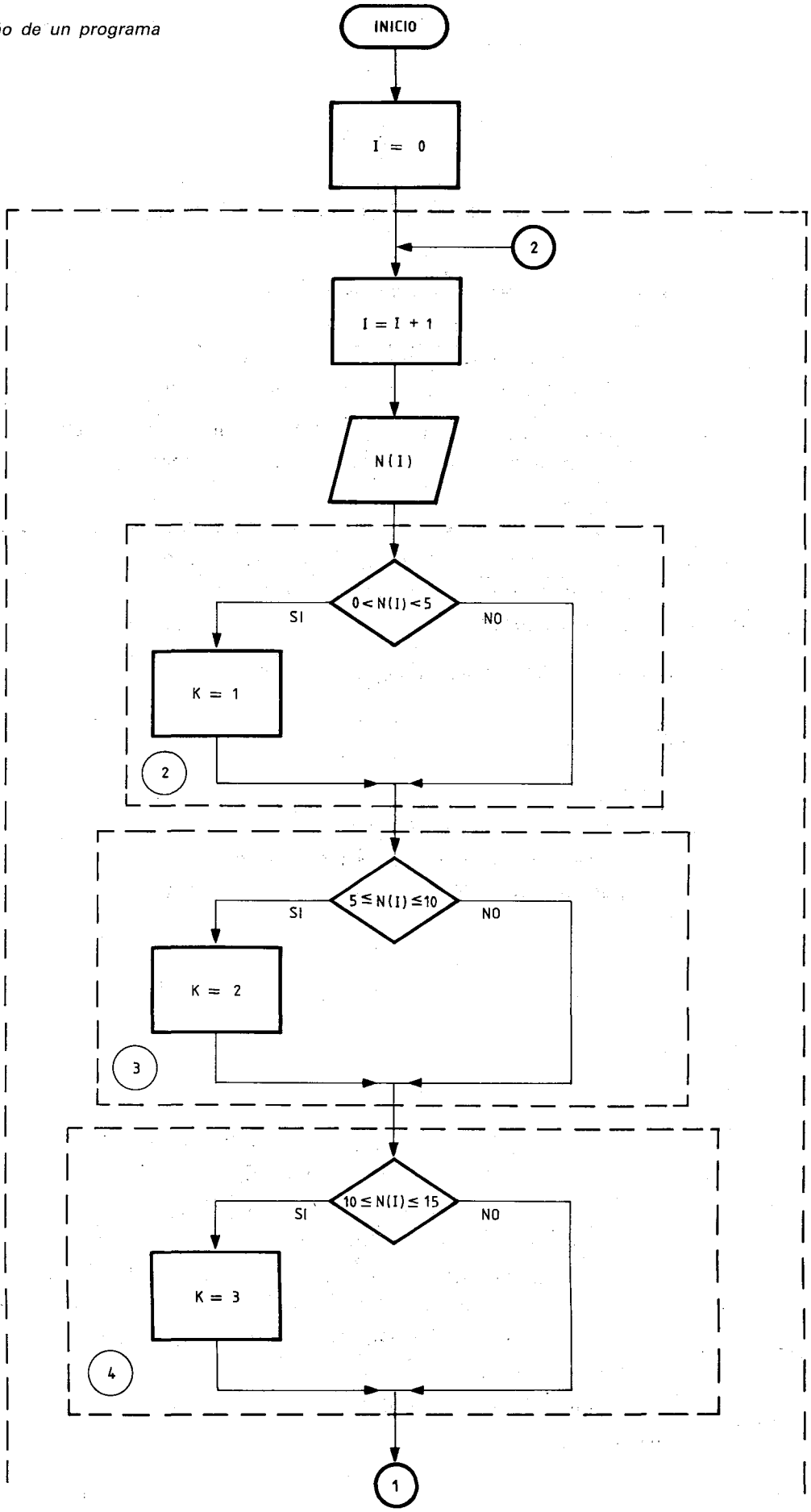
La resolución del problema es la siguiente:

Descripción Ø:

```
EMPIEZA
    Lee la lista de 1Ø números
    Verifica a qué lista, 1, 2 ó 3, corresponden
    Escribe los listados de las listas 1, 2 y 3
FINAL
```

Descripción 1:

```
EMPIEZA
    Repetir:
        Leer un número N(I).
            Si (N(I) está entre Ø y 5) hacer: K = 1.
            Si (N(I) está entre 5 y 1Ø) hacer: K = 2.
            Si (N(I) está entre 1Ø y 15) hacer: K = 3.
        Hacer (S1,S2,S3) según K:
            S1: guardar N(I) en lista 1; L(1,A) = N(I); A = A+1.
            S2: guardar N(I) en lista 2; L(2,B) = N(I); B = B+1.
            S3: guardar N(I) en lista 3; L(3,C) = N(I); C = C+1.
    Hasta: I = 1Ø.
    Escribir las tres listas; L1(A), L2(B), L3(C)
FINAL
```


INICIO
I = 0
2
I = I + 1
N(I)
0 < N(I) < 5
SI
NO
K = 1
2
5 ≤ N(I) ≤ 10
SI
NO
K = 2
3
10 ≤ N(I) ≤ 15
SI
NO
K = 3
4
1

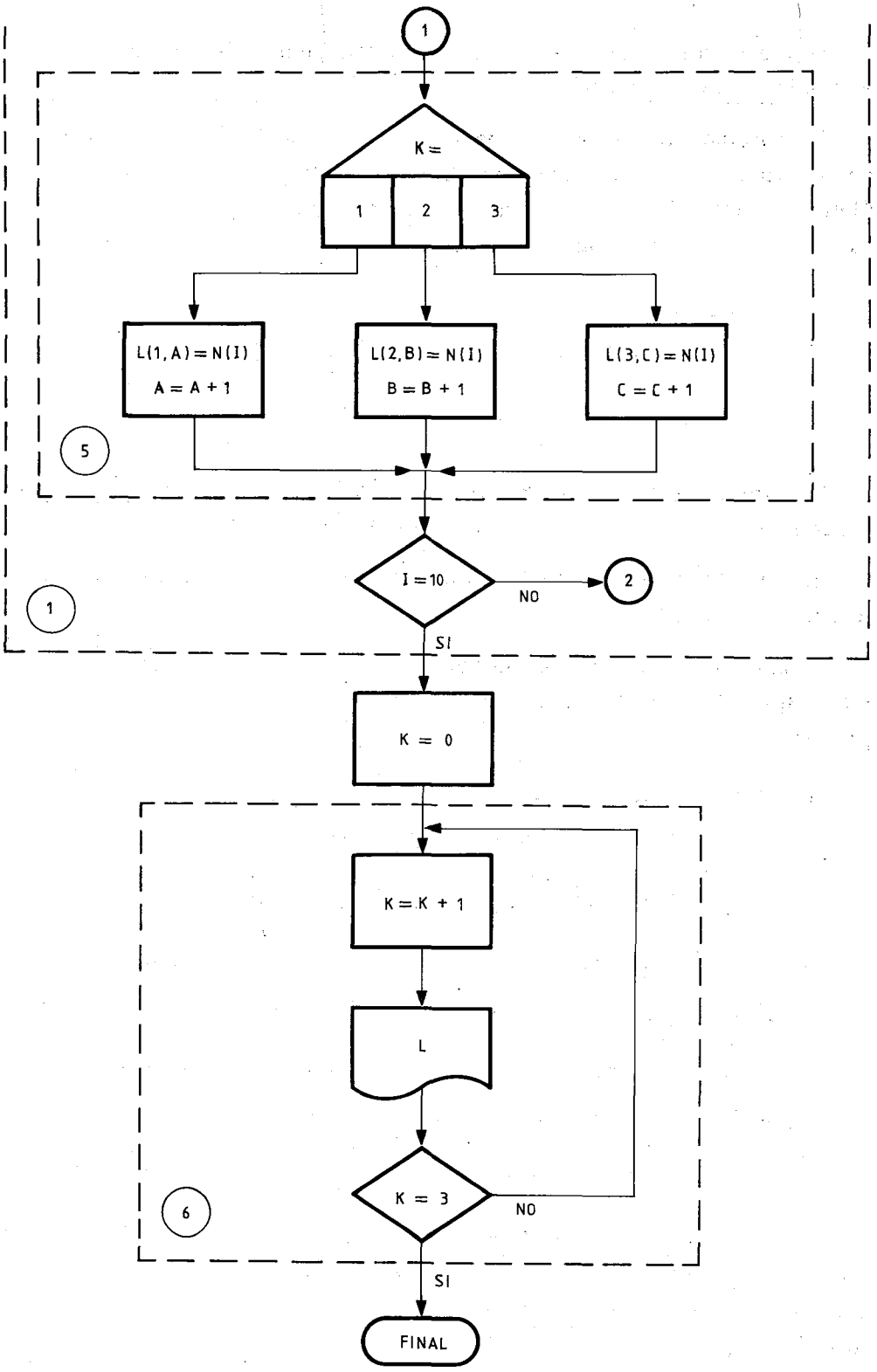



*Fig. 4.10. Colocar datos de entrada en 3 listas diferentes según su valor.*

Las estructuras usadas en estas descripciones son: la repetitiva, la alternativa y la secuencial.

El procedimiento de escribir las tres listas en el organigrama está previsto como secuencial. Primero debe escribirse una lista, luego la otra y al final la última de ellas. Podría presentarse este procedimiento o tarea según una estructura repetitiva de la forma siguiente:

*Repetir:*
escribir una lista.
*Hasta:*
que se llegue a la tercera lista.

Este modo de actuar permite codificar una sola vez la subrutina *escribir,* aplicándose sucesivamente a la primera, segunda y tercera listas.

El organigrama completo del problema se representa en el esquema de la figura 4.10 (páginas 82 y 83).

En el organigrama se representan con una línea de trazos discontínuos las diferentes estructuras usadas. Vamos a asociarlas a la «*descripción* 1» del problema:

Recuadro 1: Estructura repetitiva. Se ve al final de la misma la condición, I = 1Ø. En el cuerpo superior se ve toda la tarea, que en este caso se descompone en varias, cada una con una estructura diferente. Son las marcadas por 2,3,4,5.

Recuadro 2: Estructura alternativa con una rama en vacío.

Recuadro 3: Estructura alternativa con una rama en vacío.

Recuadro 4: Estructura alternativa con una rama en vacío.

Recuadro 5: Estructura alternativa multitarea.

Recuadro 6: Estructura repetitiva.

Las estructuras 2, 3 y 4 calculan el valor del parámetro K, que se usará en la estructura 5 como índice para la decisión. Recordemos que la condición siempre se calcula antes de la estructura y no se cambia durante la ejecución de la tarea.

La estructura 6 es la encargada de escribir las listas. La condición representada por el parámetro K va cambiando durante la ejecución de la tarea

(el valor de K se incrementa a cada paso), como corresponde a una estructura repetitiva.

Se demuestra aquí otra de las ventajas de la programación estructurada. Para poder añadir toda esa tarea de escritura en el organigrama no hay que modificarlo totalmente. Se puede coger todo el recuadro donde ahora pone *escribir L* y poner en su lugar toda la estructura repetitiva de escritura. Como esta estructura tiene una sola entrada y una única salida, como la que estaba escrita hasta ahora, la nueva estructura encaja perfectamente. Todo ello puede verse en la figura 4.11.

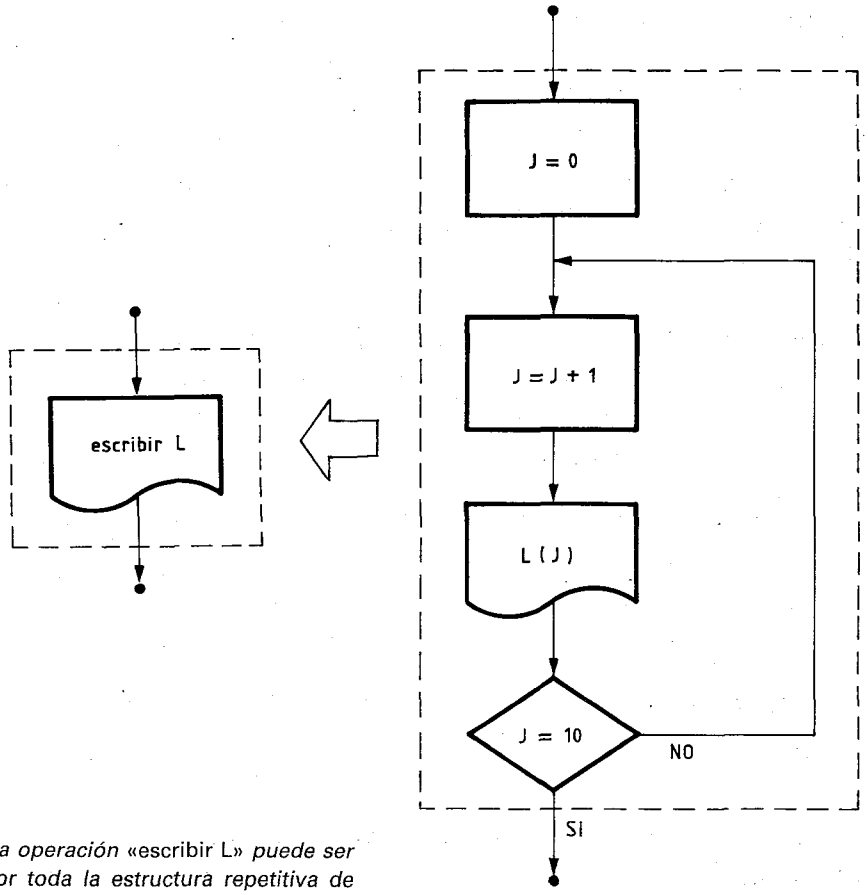



*Fig. 4.11. La operación* «escribir L» *puede ser sustituida por toda la estructura repetitiva de escritura.*

## 4.3. PRUEBAS DE MESA

Una vez se tiene la estructura general definida y se ha conseguido un organigrama en el que se hayan descompuesto las tareas al máximo, el paso que queda para concluir el diseño del programa es verificar que el programa dé los resultados buscados a partir de un juego o conjunto de pruebas correctamente diseñado.

Este conjunto de pruebas se hace sobre el organigrama y no sobre la máquina, puesto que aún no se ha codificado el organigrama. No obstante, una vez codificado el programa se tendrán que volver a realizar las pruebas y debe utilizarse para ello el mismo juego de pruebas que se usó en el diseño.

Para la realización de estas pruebas se dispone de:

— Un juego de datos de ensayo.
— Las especificaciones que debe cumplir el proceso, y
— El organigrama.

El primer punto que va a desarrollarse es pues, una construcción del juego o conjunto de datos que se usarán para la prueba. Se comentan también las características que debe cumplir este juego de datos.

La característica principal es que el juego de datos sea capaz de obtener todos los resultados solicitados por el problema. Además, y como segunda característica, debe probar el programa para cualquier tipo de datos que le aparezcan a la entrada.

Si con este juego de datos el programa diseñado (el organigrama hasta ahora) proporciona todos los resultados requeridos, la fase de diseño puede darse por terminada y se pasa a la siguiente. En caso de que los resultados no fueran satisfactorios, el programa tendría que ser revisado y corregido.

Recordemos aquí el concepto de rango o dominio de los valores de entrada de un programa. Era una noción referida a los distintos valores que pueden ponerse como entrada en un programa. Algunos de ellos daban respuesta errónea del programa. Esos valores no deben introducirse en el programa, y esa aclaración debe colocarse en la documentación del programa. Pero si el programador no está seguro de que nunca van a aparecer dichos valores, debe protegerse al programa con su presencia. El modo de efectuar esa protección se vió en el ejemplo de la resolución de la ecuación de segundo grado. Se trata de colocar, inmediatamente después de la entrada de datos, un test (o condición) que verifique si estos datos pertenecen al conjunto de datos de entrada que puede manejar el programa sin producirse error, es decir, al

dominio de valores de entrada. En caso de pertenecer a ese conjunto el proceso sigue adelante y, en caso contrario, el programa produce un mensaje de error y se para. Para la confección de las pruebas se revisa el programa desde «abajo» hasta «arriba», es decir, desde la impresión de los resultados hasta su cálculo, y, posteriormente, hasta su entrada.

Debe comprobarse si cada una de las salidas da el resultado requerido y si el cálculo se ha hecho correctamente. Después hay que ver si los bucles se ejecutan el número correcto de veces. A continuación se debe revisar si las condiciones están bien formuladas y en qué momento cambian los parámetros que hacen cumplirse la condición. Para cada uno de estos detalles claves en el programa podemos diseñar conjuntos de datos que los comprueben.

Una vez diseñado este conjunto de datos de prueba se empieza a probar el organigrama.

Se ensaya el primer juego de datos. Se repite todo el proceso con todos y cada uno de los juegos de datos que se poseen, verificando que se ha pasado por todas las ramas del organigrama, esto es, que todos hayan sido probados.

El programador debe efectuar los cálculos que pide el organigrama en sus tareas. Si es necesario, se hace uso de una calculadora de bolsillo o de mesa, que tienen un modo de actuación, en cuanto a la aritmética, semejante a la que tendrá el ordenador.

Para procurar una organización en todo este proceso es conveniente numerar todos los procedimientos y todas las ramas del organigrama, haciendo una lista del dato de entrada usado y de las ramas por las que ha pasado.

Además de esta tabla, conviene hacer otra con los nombres de todas las variables del programa, o de la parte que se vaya a probar, e ir escribiendo su valor a medida que avanza la prueba del programa. De este modo puede verse cómo se modifican los valores de esas variables y localizar el error perfectamente.

*Ejemplo*

Supongamos el problema de hacer tres listados en los que estarían los números clasificados de cero a cinco, de cinco a diez y de diez a quince.

Revisamos primero el organigrama. Vemos que existe un punto en el que se va a escribir la matriz L(K,J). Ese es el resultado. Vemos también, de la comprobación de las condiciones K y J, que la matriz tiene 3 filas y 1Ø

columnas, de modo que en cada fila se guarda una lista y en cada columna un valor de la lista.

Veamos la escritura. En la parte encargada de hacerla hay dos estructuras repetitivas, una dentro de la otra. La interior es la que escribe las columnas de uno hasta diez, mientras que la exterior es la que escribe las filas, de uno hasta tres. De acuerdo con esto, el proceso de escritura se puede dar por correcto ya que para cada una de las filas se escriben 10 columnas.

Una vez considerado el bloque de escritura, el cual ha sido decidido como correcto, se pasa al bloque siguiente, que es la formación de la matriz.

Lo que tiene que hacer este bloque es únicamente distribuir en tres partes diferentes la información que le llega. Puesto que se ha adoptado la estructura de matriz para almacenar esta información, el programa va a ir rellenando columnas de una de las tres filas, en cada caso una fila diferente.

Se comprueba que es exactamente lo que hace, puesto que el índice K se usa para rellenar la lista uno, la dos o la tres en los diferentes casos. Lo señalado en el organigrama en la parte inferior de este proceso es el incremento de los índices que señalan las columnas, necesario para poder escribir la siguiente posición.

Una vez comprobado que este proceso es correcto, se revisa el anterior, la decisión de dónde va a ser colocado un elemento. El modo de realización es dar el valor de K = 1, 2 ó 3 dependiendo del resultado de este test que verifica si el valor de entrada pertenece al rango 1, al rango 2 ó al rango 3 (de 0 a 5, de 5 a 10 y de 10 a 15 respectivamente).

Los test o condiciones están presentados como estructura alternativa que en el caso de condición afirmativa rellena el valor de K y el caso negativo no hace nada. Por lo tanto, se necesitan tres tests uno para cada rango.

Las condiciones se presentan como correctas. En la primera condición un signo mayor o igual incluye al cero como perteneciente a este rango, mientras que un signo menor deja fuera al cinco de ese primer rango. Los otros tests se expresan de la misma forma con el uso apropiado de los operadores de comparación. (Notar que la última condición incluye tanto el 10 como el 15, y por lo tanto el formato es diferente de los anteriores).

La última etapa es verificar que la lectura es correcta, notando que la condición I = 10 da una repetición desde uno hasta diez como se esperaba.

Una vez decidido que el programa efectúa correctamente los cálculos recibidos se puede empezar con la prueba del juego de datos (fig. 4.12).

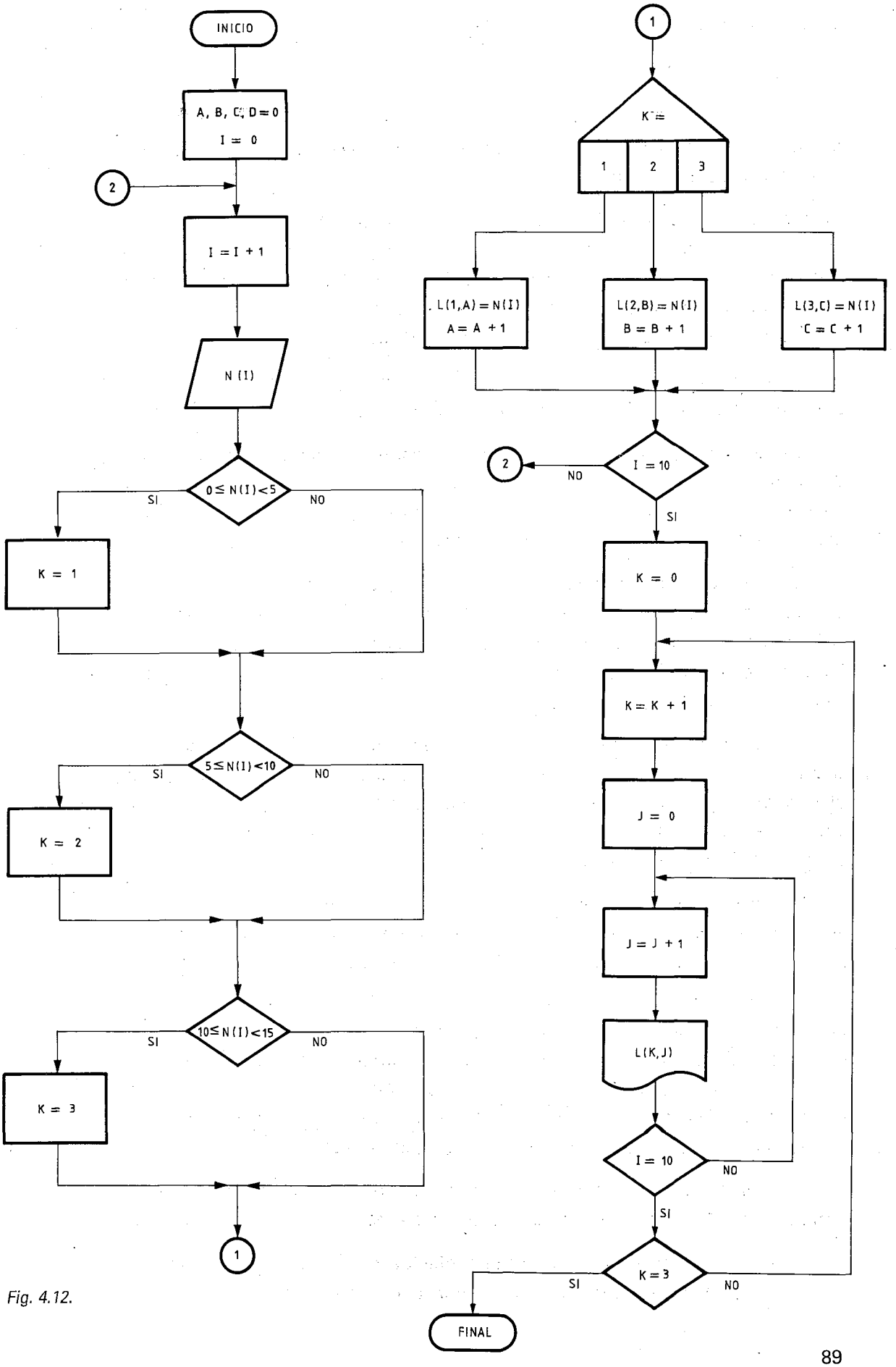

INICIO
A, B, C, D = 0
I = 0
2
I = I + 1
N (I)
0 ≤ N(I) < 5
SI
NO
K = 1
5 ≤ N(I) < 10
SI
NO
K = 2
10 ≤ N(I) < 15
SI
NO
K = 3
1
1
K =
1
2
3
L(1,A) = N(I)
A = A + 1
L(2,B) = N(I)
B = B + 1
L(3,C) = N(I)
C = C + 1
2
NO
I = 10
SI
K = 0
K = K + 1
J = 0
J = J + 1
L(K, J)
I = 10
NO
SI
K = 3
SI
NO
FINAL

*Fig. 4.12.*

El diseño del juego de datos se prepara de la siguiente forma: se presenta primero un juego de datos que no tenga ningún problema para la máquina, es decir, que la máquina los pueda clasificar *sin estar al borde de la condición.* Estos datos pueden ser: 3, 6, y 12, que serían clasificados respectivamente en cada una de las tres listas. Se le presentan a continuación un juego de datos en el borde de la condición: 0, 5, 10 y 15, los cuales deben probar el funcionamiento correcto de las condiciones.

Por último, queda por probar el bucle de lectura. La forma de hacer esta prueba es suponer que ya se ha llegado a la iteración número 8 y ponerle 3 datos más para que calcule (debe aceptar los dos siguientes y rechazar el tercero).

Las tablas que se necesitan para la corrección pueden tener la forma siguiente:

| *Juego de datos* | *Estructura probada* |
|---|---|
| 3, 6, 12 | Cálculo y escritura |
| Ø, 5, 1Ø, 15 | Condiciones de cálculo y escritura |
| 9, 11, 12 | Bucle de lectura |

*Prueba del primer juego: 3, 6, 12*

La tabla de valores de las variables para esta prueba es:

| Entrada | I | N(I) | K | A | B | C | A<br>B<br>L(K,C) | J |
|---|---|---|---|---|---|---|---|---|
| 3 | Ø<br>1 | 3 | 1 | 1<br>2 | 1 | 1 | 3 | Ø |
| 6 | 2 | 6 | 2 | 2 | 1<br>2 | 1 | 6 | Ø |
| 12 | 3 | 12 | 3 | 2 | 2 | 1<br>2 | 12 | Ø |

El bucle de escritura no se ha presentado en la tabla. Se ve que mientras el valor de K se mantiene fijo en 1, el valor de J va variando de 1 a 1Ø, escribiéndose toda la fila, que en este caso estará compuesta por un 3 seguido de nueve ceros.

Seguidamente K cambia al valor 2 mientras que J varía de 1 a 1Ø escribiéndose, igual que antes, un 6 seguido de nueve ceros.

Con K = 3 se tendría, del mismo modo, un 12 seguido de nueve ceros.

Las pruebas siguientes darían como resultado una tabla semejante.

En ningún caso se llegaría a la escritura porque mientras I no llegara al valor 1Ø el programa no saldría del bucle de cálculo.

## EJERCICIOS

1. Las ventajas de la programación modular son:

   *a*) No tiene ventajas porque descompone mucho el problema.
   *b*) Puede hacerse por varias personas a la vez.

2. Una de las características de la programación modular es:

   *a*) En cada paso el problema está resuelto completamente.
   *b*) Debemos llegar al final del programa para tener el problema resuelto.

3. La programación estructurada consigue

   *a*) La normalización en la programación.
   *b*) Las ventajas de la programación modular.
   *c*) Ambas a la vez.

4. La metodología de "arriba-abajo".
   *a)* Permite el desarrollo del problema paulatinamente, de lo general a lo particular.
   *b)* Desarrolla el problema paulatinamente, de lo particular a lo general.

5. La conexión entre cada una de las tareas en que se divide un programa se resuelve:

   *a)* Mediante entradas y salidas de información.
   *b)* Por subrutinas.

6. En la estructura repetitiva:

   *a)* Una condición indica el número de veces que se ejecuta una tarea.
   *b)* Se prueba varias veces la condición, hasta que se cumple.

7. En la estructura repetitiva:

   *a)* La condición varía en cada bucle.
   *b)* La condición no cambia, son los valores de los parámetros que expresan la condición los que cambian.

8. En la estructura alternativa

   *a)* Se ejecuta una sola de las tareas.
   *b)* Los parámetros de la condición cambian en un paso anterior del programa.
   *c)* Ambas a la vez.

9. La multitarea es:

   *a)* Un caso particular de la estructura secuencial.
   *b)* Un caso particular de la estructura repetitiva.
   *c)* Un caso particular de la estructura alternativa.

# 5.

# CODIFICACION

## CUADRO RESUMEN DE LA LECCION

- **Codificación**
  - — Subprogramas
  - — Codificación de las estructuras
    - — de la estructura secuencial.
    - — de la estructura repetitiva.
    - — de la estructura alternativa.
  - — Estructuras de datos
    - — Vectores y matrices
  - — Tipos de datos
  - — Errores
    - — Construcción errónea de operaciones
    - — Mezcla de variables de distinto tipo
    - — Uso indebido de la aritmética de la máquina

En este capítulo se expondrán todas las herramientas que se tienen para la codificación del organigrama diseñado en el capítulo anterior, según las normas de estructuración comentadas en él. Se describirán, pues, las bases de datos que poseen los lenguajes y las estructuras de instrucciones que tienen dichos lenguajes. De este modo podremos adaptar cada una de las estructuras a cada uno de los casos presentados en el capítulo anterior.

Por último se analizarán las posibles causas de error que supone una codificación deficiente. No debe olvidarse que la codificación es un paso sin ninguna dificultad si se posee un eficaz método.

## 5.1. SUBPROGRAMAS

Un subprograma es un conjunto de instrucciones dedicadas a la resolución de una tarea y que forma parte de un programa principal. El programa principal es el encargado del control y de las decisiones; cuando es necesaria la ejecución de una tarea particular se encarga dicha ejecución a un subprograma.

Como el programa está estructurado en bloques, cada bloque puede ser una subrutina viéndose perfectamente las partes de que consta el programa y lo que realiza. De este modo se gana claridad en la programación.

Esta utilización de las subrutinas como componentes de programas es un avance respecto al uso tradicional. Allí se recomendaba el uso de las subrutinas cuando un proceso del programa principal tenía que ejecutarse más de una vez en diferentes puntos de programa. Se decía que no tenía ningún sentido escribir el proceso varias veces en el programa principal cuando podía ser

escrito una sola vez como subrutina y ser llamado las veces que fuera necesario desde este programa principal. Ahora vemos que, aunque sólo se utilice una vez, una tarea puede ser descrita como subrutina.

*Ejemplo*

Se trata de leer una lista alfabética que expresa una operación aritmética básica, y dar los datos necesarios para su ejecución.

La lista alfabética tendrá tres caracteres: dos operandos y el signo de la operación elegido entre los cuatro básicos: +, −, ×, /

Un ejemplo de entrada puede ser: A − B, 10, 12

A lo que el ordenador tendría que dar como salida

A = 10
B = 12
A − B = − 2

El ordenador habría identificado la operación a realizar: la resta en este caso. Habría identificado los operandos: A = 10 y B = 12. Y habría calculado el resultado de la operación: −2.

El procedimiento para la realización de este problema se muestra en los pasos siguientes, haciendo uso de los conocimientos de la programación estructurada.

Descripción Ø: Subrutina ENTRA

EMPIEZA
    Identificar la operación y calcular el resultado.
FINAL

Descripción 1:

EMPIEZA
    Entrar los datos
    Identificar la operación y calcular resultado
    Dar la salida.
FINAL

La descripción debe avanzar más para poder resolver el problema, aunque ya se puede escribir un programa según la descripción 1:

```
1Ø  Llamar a subrutina ENTRA
2Ø  Llamar a subrutina CALCULO
3Ø  Llamar a subrutina SALIDA
4Ø  Fin
```

Como puede verse, el programa es completo y para su realización sólo es necesario el desarrollo de las subrutinas, lo que será motivo de sucesivas descripciones. Cada una de ellas puede tratarse por separado.

*Estudio de la subrutina ENTRA*

La estructura de datos que se adoptará es un vector alfanumérico que recibirá la expresión. Tendrá tres posiciones, una para cada carácter de la expresión. Además, poseerá otro vector de dos posiciones para recibir los operandos. Este vector es numérico. La descripción es la siguiente:

Descripción 0:

```
EMPIEZA
    Entrar expresión
    Entrar operandos
FINAL
```

Descripción 1:

```
EMPIEZA
    Repetir:
        Entrar carácter.
    Hasta:
        Que hayan entrado 3 caracteres.
    Repetir:
        Entrar dato numérico.
    Hasta:
        Que hayan entrado 2.
FINAL
```

El organigrama correspondiente se muestra en la figura 5.1.

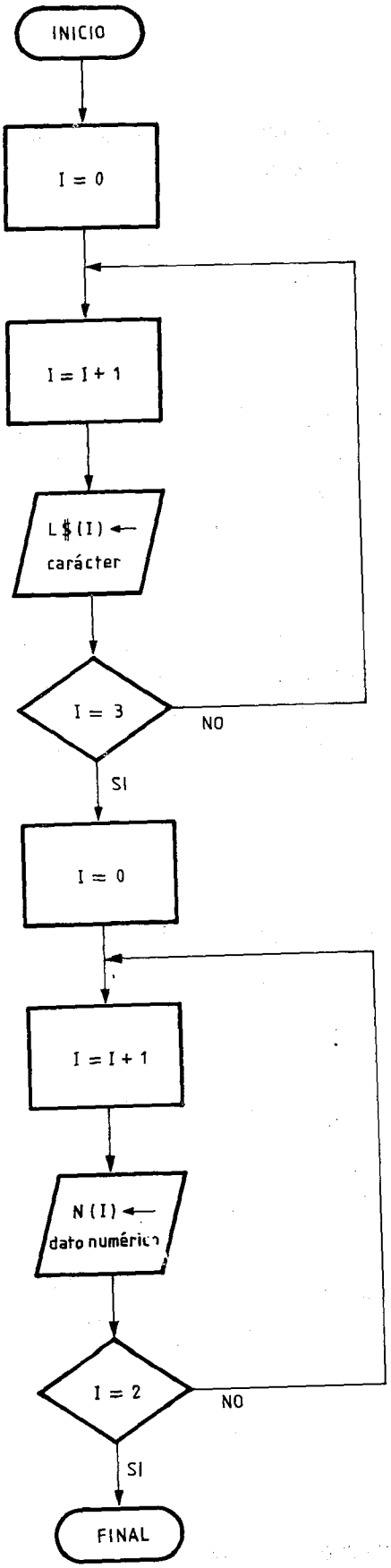



*Fig. 5.1. Interpretación de una expresión: Subrutina ENTRA.*

Al finalizar esta subrutina se tiene un vector, L$(I), en el que se ha almacenado la expresión y otro vector, N(I), en el que se han almacenado los datos.

*Estudio de la subrutina CALCULO*

La subrutina va a estudiar el carácter L$(2) y hará los diferentes cálculos dependientes de su valor.

La descripción es la siguiente:

Descripción ∅: Subrutina CALCULO

```
EMPIEZA
    Si L$(2) = "+" entonces K = 1.
    Si L$(2) = "-" entonces K = 2.
    Si L$(2) = "*" entonces K = 3.
    Si L$(2) = "/"  entonces K = 4.
    Hacer (S1, S2, S3, S4) según K.
        S1:
            S = N(1) + N(2); S$ = «SUMA».
        S2:
            S = N(1) - N(2); S$ = «RESTA».
        S3:
            S = N(1) * N(2); S$ = «PRODUCTO».
        S4:
            Sí N(2) = ∅ entonces S$ = «ERROR».
            en caso contrario: S = N(1)/N(2); S$ = «DIVISION».
FINAL
```

Se aprecia el uso de la estructura multitarea para decidir qué operación se tiene que realizar.

La subrutina CALCULO rellena la salida. En primer lugar ha utilizado una variable que se encarga de almacenar el resultado, S. Hace uso también de otra variable, ahora alfanumérica (S$), que describe la operación efectuada. Para el caso de la división verifica el caso en que el divisor es cero, lo que daría error en la máquina si se realizara la operación.

El organigrama correspondiente se muestra en la figura 5.2.

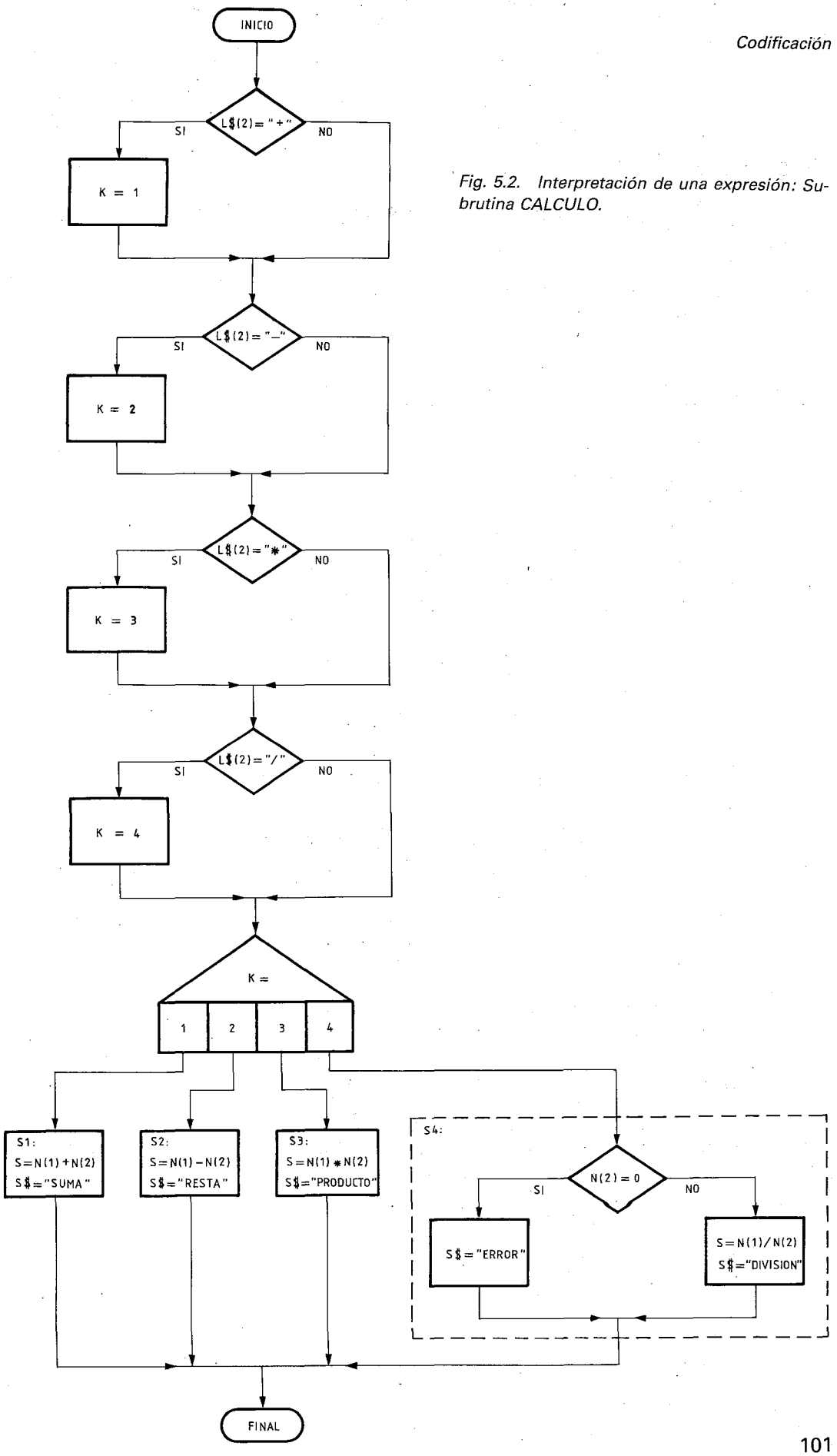



*Fig. 5.2. Interpretación de una expresión: Subrutina CALCULO.*

Queda por último desarrollar de la subrutina SALIDA.

Descripción Ø: Subrutina SALIDA

```
EMPIEZA
    Sacar comentarios.
    Sacar resultados.
FINAL
```

Descripción 1:

```
EMPIEZA
    Escribir
        «LOS DATOS SON:»
        L$(1) " = " N(1).
        L$(3) " = " N(2).
        «LA OPERACION ES».
        S$.
    Repetir:
        Escribe L$(I)
    Hasta:
        Que se haya escrito 3 veces.
    Escribir " = ".
    Escribir S.
FINAL
```

Según esta subrutina el formato de salida para los datos de entrada: A + B, 1, 1, será:

```
LOS DATOS SON:
    A = 1
    B = 1
LA OPERACION ES:
    SUMA
    A + B = 2
```

El organigrama se presenta en la figura 5.3.

El punto y coma (;) usado en alguna ocasión en el organigrama indica que el carro de la impresora no vuelve y que lo próximo que se escribe se hace en la misma línea.

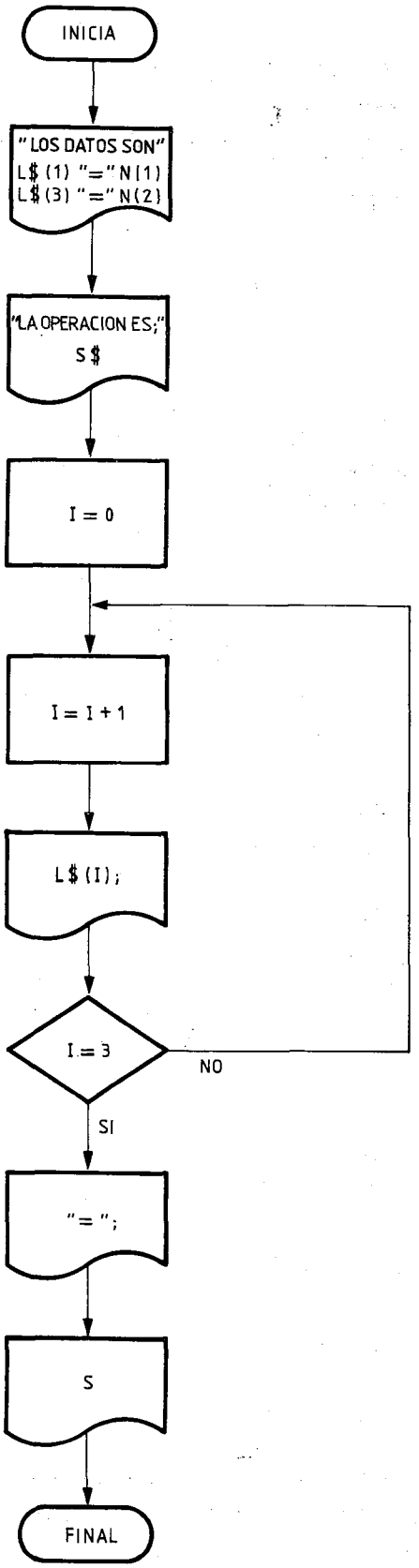



*Fig. 5.3. Interpretación de una expresión: Subrutina SALIDA.*

Los programas correspondientes a cada una de estas subrutinas se muestran en las líneas siguientes:

*Subrutina ENTRA*

```
  Inicializar I (I = 0)
(1) Incrementar I
    Leer el caracter y colocarlo en el vector L$(I)
    Test de I: ¿I = 3?
          Sí, continuar el proceso
          No, volver a la instrucción (1)
    Inicializar I (I = 0)
(2) Incrementar I
      Leer dato y almacenar en vector N(I)
      Test de I ¿Es I = 2?
          Sí, devolver el control al programa principal
          No, volver a la instrucción (2)
```

*Subrutina CALCULO*

```
    Leer el carácter L$(2)
    ¿Es un + ?
        Sí, hacer K = 1
        No, continuar
    ¿Es un − ?
        Sí, hacer K = 2
        No, continuar
        ¿Es un *?
        Sí, hacer K = 3
        No, continuar
    ¿Es un / ?
        Sí, hacer K = 4
        No, continuar
        Hacer S1, S2, S3 ó S4 según el valor de K (K = 1, 2, 3 ó 4)
        S1:
            Hacer S = N(1) + N(2)
            Hacer S$ = «SUMA»
```

```
S2:
    Hacer S = N(1) – N(2)
    Hacer S$ = «RESTA»
S3:
    Hacer S = N(1) * N(2)
    Hacer S$ = «PRODUCTO»
S4:
    Es N(2) = Ø?
      Sí, hacer S$ = «ERROR»
      No, hacer S$ = «DIVISION»
          hacer S = N(1)/N(2)
Devolver el control al programa principal
```

*Subrutina SALIDA*

```
    Escribir «LOS DATOS SON»
    Escribir L$(1) " = " N(1)
    Escribir L$(3) " = " N(2)
    Escribir "LA OPERACION ES":
    Escribir S$
    Inicializar I (I = 0)
(1) Incrementar I
    Escribir L$(I);
    ¿Es I = 3?
    Sí, continuar
    No, volver a instrucción (1)
    Escribir "=" ;
    Escribir "S"
    Devolver el control al programa principal.
```

Este ejemplo ha servido para ver el uso de las subrutinas en la confección de programas estructurados. Recordemos que una subrutina puede llamar a otra subrutina. Si bien en este ejercicio no se ha hecho uso de esta característica porque el proceso era muy sencillo, es conveniente tenerlo en cuenta para problemas más complicados. Un lugar apropiado para el uso de esas subrutinas llamadas desde otras subrutinas hubiera sido en cada una de las tareas S1, S2, S3, S4 de la subrutina CALCULA.

## 5.2. CODIFICACION DE LAS ESTRUCTURAS

La potencia de un ordenador se basa en su facultad de ejecutar programas almacenados y de tomar decisiones. Esto no quiere decir que el ordenador sea capaz de pensar sino que puede romper la secuencia del programa en función de una condición.

Además de esta posibilidad el ordenador es capaz de manejar datos y hacer operaciones; de modo que las instrucciones que puedan ejecutarse estarán clasificadas según esas funciones, para cualquier lenguaje que se utilice. Estas son:

— Instrucciones de control y decisión.
— Instrucciones de operación o procesamiento.
— Instrucciones de entrada y salida o de intercambio de información

Si se observa un organigrama se ve que cualquiera de los símbolos que allí se expresan tienen un tipo de instrucción equivalente.

Pueden encontrarse rombos que se implementan con una instrucción de control, cuadrados que se implementan con instrucciones de procesamiento y romboides que se implementan con instrucciones de entrada y salida.

Todo esto en cuanto a la llamada programación natural. Ahora sería conveniente encontrar instrucciones que implementaran las estructuras que se han venido desarrollando ya que se ha decidido que la programación estructurada tenía ventajas sobre la natural.

Existen algunos lenguajes que están más cerca de la programación estructurada que otros. Esto significa que tienen instrucciones que se acoplan directamente a cada una de las estructuras. Un ejemplo de ellos sería el ALGOL. En general, los lenguajes de programación que se están desarrollando actualmente tienen en cuenta este factor y ya organizan sus instrucciones en base a esas estructuras.

Se repasan ahora las estructuras que se irán codificando mediante el lenguaje BASIC.

### 5.2.1. Codificación de la estructura secuencial

La idea de secuencia en el programa BASIC la da la numeración de las instrucciones, ejecutándose una detrás de otra, mientras no sean de ruptura de secuencia (bifurcación). En otros lenguajes es tan sólo el orden de escritura.

*Ejemplo:*

```
1Ø LET A = Ø
15 LET B = 1
2Ø LET C = A+B
```

En este caso se ejecuta una instrucción tras otra desde la 1Ø hasta la 2Ø.

### 5.2.2. Codificación de la estructura alternativa

En este caso existe parte de condición y parte de tarea, luego las instrucciones a usarse serán las de control (para la condición) y las de procesamiento (para la tarea).

*Ejemplo:*

Usaremos el caso ya estudiado de verificar cuál es el mayor de dos números. Supongamos que se ha verificado anteriormente que los números no son iguales y sólo se codifica la estructura alternativa.

El programa es el siguiente:

```
1Ø IF A > B THEN M = A
2Ø GOTO 3Ø
25 M = B
3Ø...
```

En este caso los números a comparar se han llamado A y B. M es la variable donde se coloca el más grande. La instrucción 2Ø es necesaria para que no se mezcle una rama con otra; ya que las dos tareas de la estructura alternativa son excluyentes, y sólo se tiene que realizar una de las dos en cada caso.

Algunas versiones del lenguaje BASIC no permiten en la introducción de control un procesamiento, sino sólo el número de la próxima instrucción a ejecutar.

### 5.2.3. Codificación de las estructuras repetitivas

Para este tipo de estructuras los lenguajes suelen poseer instrucciones que ejecutan un bucle. Todo lo que está en el interior del bucle se ejecuta repetidamente un número de veces indicado en la definición del bucle.

Esta estructura puede ejecutarse reuniendo en el programa instrucciones de decisión y de procesamiento.

*Ejemplo:*

Escribir los 10 primeros números naturales.

```
1Ø  FOR I = 1 TO 1Ø
2Ø  PRINT I
3Ø  NEXT I
```

Donde la instrucción 2Ø se repite diez veces y el bucle está descrito mediante las instrucciones 1Ø y 3Ø.

## 5.3. ESTRUCTURAS DE DATOS

El programa debe tener en cuenta, para su realización correcta, las estructuras de datos de entrada y de salida, así como la estructura del programa. Una vez estudiadas las estructuras del programa, veamos las estructuras de datos que poseen normalmente los lenguajes.

### 5.3.1. Vectores y matrices

Los *vectores* y *matrices*, son referidos habitualmente en los lenguajes como variables con subíndices. El número de subíndices indica si se trata de un caso o de otro. Se reserva el nombre de *vectores* para las variables con *un* subíndice y el de *matrices* para las de *varios* subíndices (estrictamente, una matriz se representaría únicamente por variables de dos subíndices, pero se generaliza a más de dos porque algunos lenguajes lo permiten).

El número de elementos máximo que va a poder contener la variable se expresa mediante la dimensión de esta variable. La dimensión *uno* en vector coincidiría con la variable sin subíndices.

Para que la codificación también pueda desarrollarse independientemente por medio de subprogramas sólo se ha de definir cuál es la estructura de datos que pasará una estructura de información a la siguiente estructura. Ese es el nexo de unión que tiene una estructura con otra.

## 5.4. TIPOS DE DATOS

Un programa puede manejar distintos tipos de datos:

*a)* Números enteros (o de punto fijo)
*b)* Números reales (o de punto flotante)
*c)* Lógicos
*d)* Alfanuméricos
*e)* Números reales de doble precisión
*f)* Números complejos

Estos dos últimos son usados, sobre todo, en lenguajes dedicados al cálculo. Por ejemplo, el de doble precisión sirve para representar números en los ordenadores con mayor precisión de la usual, lo que significa que la representación ocupa más memoria. Los complejos son una pareja de números reales, es decir, como una variable de dimensión dos. La máquina entiende que tiene que reservar dos posiciones para su almacenamiento, una para cada valor de la pareja.

En programación se distingue la utilización de los números enteros y la de los números reales.

Los números *enteros* se representan con fidelidad absoluta en las calculadoras, siempre que su valor se mantenga dentro del límite que puede soportar la calculadora. Son, pues, convenientes para la implementación de las condiciones con el signo igual o para usarse como *contadores*.

Los contadores son los sistemas usados para verificar cuántas veces se ha repetido una tarea. El contador se inicializa a 0, la primera vez que se va a ejecutar la tarea, y se va incrementando en una unidad cada vez que la tarea se ha realizado. Al final de la repetición, una lectura del contenido del contador nos da la idea exacta de las veces que se ha repetido dicha tarea.

Los números enteros son también adecuados para usarse en la confección de estructuras repetitivas cuando un lenguaje no tenga instrucciones para ejecutar bucles.

Debe huirse de las comparaciones en los controles usando el signo igual, a menos que se comparen números enteros, variables o constantes lógicas o alfanuméricas, que tienen representación exacta en la máquina. Así, siempre que se efectúa una condición en la que intervengan números reales, que la máquina no puede representar exactamente, debe usarse cualquier signo de comparación menos el signo igual, el cual, de utilizarse, llevaría consigo el que posiblemente nunca se cumpla la condición.

El cero es un número difícilmente representable como dato real. Una condición en que intervenga el cero también se tendrá que rechazar. En su lugar se podrá decir:

«repetir un cálculo hasta que el resultado sea un valor muy pequeño, $10^{-12}$ por ejemplo»,

pero nunca:

«repetir un cálculo hasta que el resultado sea cero».

Porque posiblemente la máquina entrará en un bucle del que nunca saldrá por no cumplirse la condición.

*Ejemplo: Cálculo de la raíz cuadrada de un número positivo*

En realidad ninguna máquina posee como operación la raíz cuadrada, lo que ocurre es que algunas máquinas poseen una subrutina interna de función.

Al encontrar en el programa las siglas que expresan el código de operación de la raíz cuadrada (habitualmente SQRT), el programa llama a la subrutina y le entrega el valor radicando de esa raíz. La subrutina calcula el valor de esa raíz y devuelve el control al programa principal, entregándole el resultado.

Se desarrolla a continuación un programa para el cálculo de raíces cuadradas.

Un número puede escribirse como producto de otros dos

$$A = B * C$$

Si los dos factores son iguales (B = C) cualquiera de ellos será exactamente la raíz cuadrada de A.

$$A = \sqrt{A} * \sqrt{A} = B * C$$

El algoritmo empleado elige dos números diferentes que multiplicados den el radicando. Luego va cambiando estos números hasta que se hagan iguales. En este caso ya se ha hallado la solución, cualquiera de estos números es la raíz cuadrada.

Los dos números escogidos para el comienzo son: uno cualquiera G, y el siguiente viene definido por la condición de que el producto de ambos sea el radicando. Es decir, el segundo factor es A/G.

Hasta ahora sabemos que la raíz está comprendida entre esos dos números (Fig. 5.4).

Ahora se acerca sucesivamente G hacia A/G. El algoritmo acabará cuando G' y A/G' coincidan.

El siguiente valor de G se toma en la mitad de la distancia (D) entre G y A/G. El nuevo valor de G' es:

$$G' = G + D/2$$

A partir de ese valor de G' se vuelve a calcular el otro valor de A/G'.

Debe decidirse si se sigue conservando la ordenación de que G se encuentre a la izquierda y que G/A se encuentre a su derecha. En caso contrario cambiarían los papeles de esos dos valores. Hay que tener en cuenta que siempre se ha tomado como generador el que está a la izquierda.

Todo este proceso se repite hasta que se cumpla la condición prevista, es decir,

$$G = A/G$$

Puesto que G y A/G son valores reales, la condición no puede expresarse de esa manera, porque tiene un signo igual.

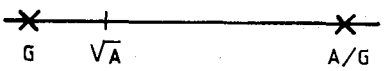



*Fig. 5.4. Cálculo de la raíz cuadrada. Disposición inicial.*

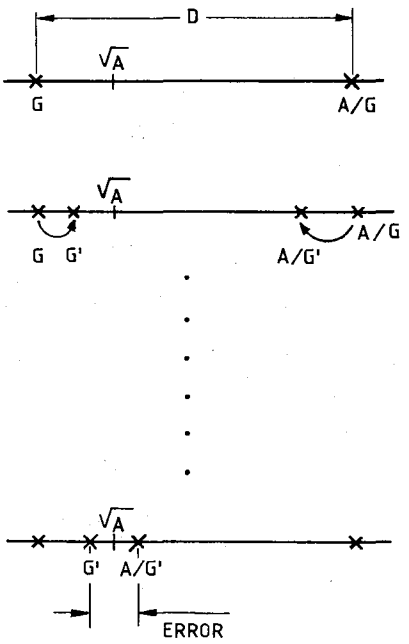



*Fig. 5.5. Proceso de aproximación a la raíz cuadrada.*

Si se obtiene sólo una aproximación de la raíz cuadrada, no debe restarse credibilidad a la computadora ya que la mayoría de raíces darían valores irracionales que matemáticamente no tienen una representación finita.

La aproximación consistirá en que la diferencia entre los dos factores sea menor que una cierta cantidad (llamada ERROR), pero no cero.

El proceso se muestra gráficamente en la figura 5.5.

Descripción ∅:

EMPIEZA
    calcular la raíz cuadrada de un número positivo.
FINAL

Descripción 1:

EMPIEZA
*Repetir:*
Calcular factores.
Verificar su posición.
*Hasta:*
Distancia entre factores sea menor que ERROR.
Escribir resultado.
FINAL

Descripción 2:

EMPIEZA
Inicializaciones, G = 1; D = 0; B = 1.
Lectura del radicando, A.
*Repetir:*
D = G − B.
G = G + D/2.
B = A/G.
Si G<B *entonces:* continuar
*En caso contrario:*
cambiar G por B; G⇆B.
*Hasta:*
|G − B|< ERROR.
Escribir G.
FINAL

Ahora debe verse cómo se hace el cambio de G por B.

Las variables G y B expresan una posición de memoria. Por la forma de trabajar de la máquina, si se escribe un dato en una posición de memoria el contenido de esa posición se borra cuando se almacena un nuevo dato. Por lo tanto, *no se puede escribir ese cambio G por B de la forma siguiente:*

```
1Ø  LET C = G
2Ø  LET G = B
3Ø  LET B = C
```

En la instrucción 3Ø se ha usado el valor anterior de G, almacenado para su conservación en C.

El organigrama se muestra en la figura 5.6. En él se ha hecho uso de las barras verticales, que, cuando están limitando a una expresión, significan valor absoluto.

La codificación del problema es:

```
1Ø  INPUT A
2Ø  G = 1; D = Ø; B = 1
3Ø  D = ABS (G – B); G = G + D/2; B = A/G
4Ø  IF (G<B) THEN 5Ø
45  C = G; G = B; B = C
5Ø  IF (B – G > = 1.E – 12)  THEN 3Ø
6Ø  PRINT G
7Ø  STOP: END
```

Al efectuar la codificación se ha empleado la función de ABS (X) que calcula el valor absoluto de una expresión, es decir, no tiene en cuenta el signo de esa expresión, ya que para calcular la distancia entre G y B no es necesario conocer cuál está a la izquierda (menor) y cuál a la derecha (mayor) sino sólo la separación entre las dos. El error se ha decidido en principio que sea del orden de $10^{-12}$ que representado como constante real es 1E – 12.

Se muestra a continuación algunos ejemplos de ejecución del programa para que pueda verse la duración del mismo y las aproximaciones que efectúa hasta que llega al resultado.

*Ejemplo* 1:

Dato: A = 2

El ciclo de aproximación es:

| Ciclo | G | B |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 1,5 | 1,3333333 |
| 3 | 1,4166667 | 1,41176467 |
| 4 | 1,41421568 | 1,41421144 |
| 5 | 1,41421356 | 1,41421356 |

Resultado = 1.41421356
SQR = 1.41421356.

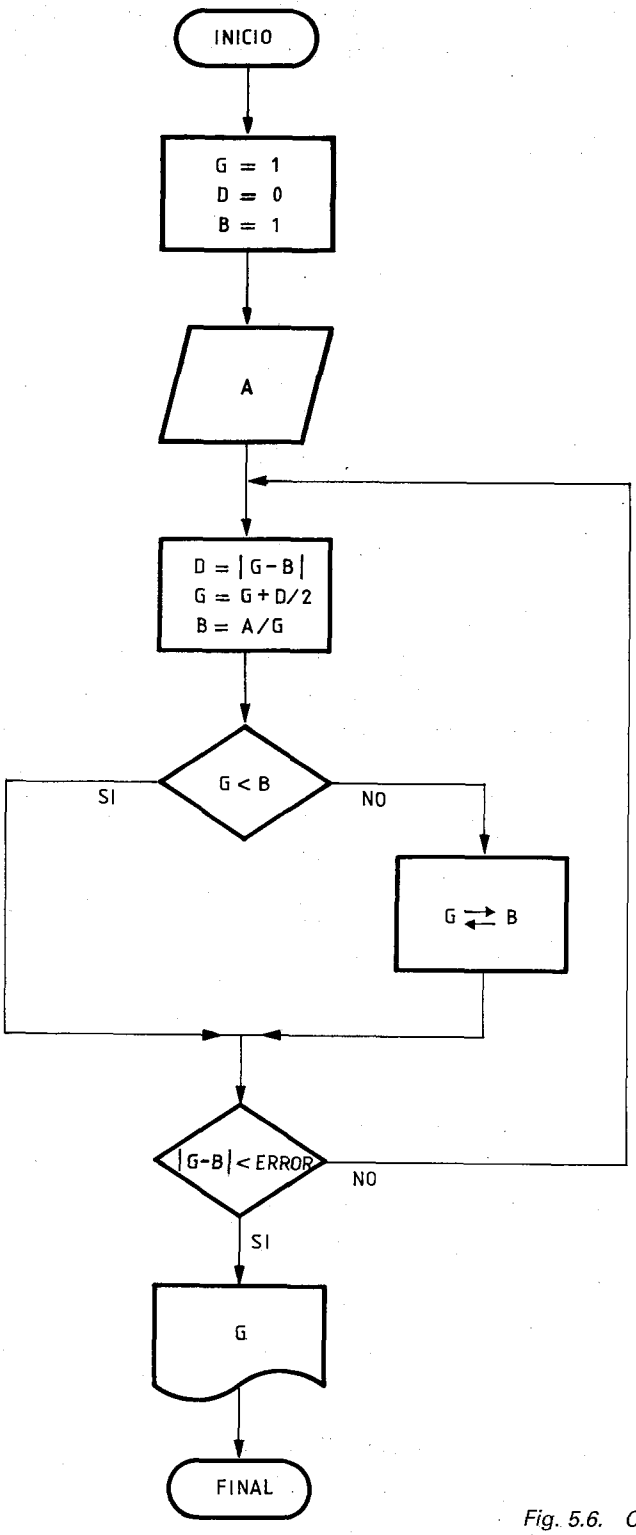



*Fig. 5.6. Cálculo de la raíz cuadrada.*

*Ejemplo* 2:

Dato A = 49

Las aproximaciones son:

| Ciclos | G | B |
|---|---|---|
| 1 | 1 | 49 |
| 2 | 25 | 1,96 |
| 3 | 13,48 | 3,635Ø1483 |
| 4 | 8,5575Ø742 | 5,7259664Ø |
| 5 | 7,14173691 | 6,861Ø76Ø4 |
| 6 | 7,ØØ14Ø647 | 6,99859381 |
| 7 | 7,ØØØØØØ14 | 6,99999986 |
| 8 | 7 | 7 |

Resultado = 7
SQR = 7

*Ejemplo* 3:

Dato: $A = 10^6$

Las aproximaciones dan:

| Ciclo | G | B |
|---|---|---|
| 1 | 1 | 1ØØØØØØ |
| 2 | 5ØØØØØ,5 | 1,999998 |
| 3 | 25ØØØ1,25 | 3,99998 |
| 4 | 125ØØ2,625 | 7,999832ØØ |
| 5 | 625Ø5,3124 | 15,9986Ø1 |
| 6 | 3126Ø,6555 | 31,989Ø924 |
| 7 | 15646,3223 | 63,9127828 |
| 8 | 7855,11754 | 127,3Ø5542 |
| 9 | 3991,21155 | 25Ø,55Ø487 |
| 10 | 212Ø,881Ø2 | 471,5Ø2168 |
| 11 | 1296,1916Ø | 771,49Ø881 |
| 12 | 1Ø33,84125 | 967,266492 |
| 13 | 1ØØØ,55385 | 999,446456 |
| 14 | 1ØØØ,ØØØ16 | 999,99984Ø |
| 15 | 1ØØØ | 1ØØØ |

Resultado = 1ØØØ.
SQR = 1ØØØ.

De los resultados anteriores se puede ver que el programa tarda más en converger a una solución, según el número radicando es mayor. Puede llegar un momento en que, el número sea demasiado grande o la condición demasiado estricta y entonces el bucle se repetiría demasiadas veces (demasiados ciclos). Una forma de evitar esto es poner un contador de bucle y decidir la finalización del bucle cuando el contador llegue a una posición determinada, por ejemplo, 100 iteraciones.

## EJERCICIOS

1. ¿Puede realizarse un programa enteramente con subprogramas?

   *a*) Sí, siempre que esté estructurado.
   *b*) No, porque el subprograma sólo se utiliza cuando una tarea va a repetirse más de una vez.

2. ¿Reconoce el tipo de estructura utilizado en la figura 4.1 (página 74) para la salida de resultados?

   *a*) Alternativa.
   *b*) Secuencial.

3. En la figura 5.2 se presenta la subrutina CALCULO. ¿Reconoce la estructura con qué finaliza el organigrama?

   *a*) Secuencial
   *b*) Alternativa

4. ¿Usaría en algún momento una instrucción de control para realizar una estructura secuencial?

   *a*) Sí, porque tiene que controlar cuándo se acaba la secuencia.
   *b*) No.

5. Una instrucción de procesamiento. ¿Puede usarse en cualquier estructura?

   *a*) No, sólo en las secuenciales.
   *b*) Sí, porque en cualquiera de ellas hay algún proceso que efectuar.
   *c*) Sólo en las alternativas y secuenciales que tienen tareas que resolver.

6. ¿Una constante alfanumérica puede almacenarse en un vector?

   *a*) Sólo cuando éste es numérico.
   *b*) No, porque en los vectores sólo se almacenan variables.
   *c*) Sí, siempre que sea alfanumérico.

7. ¿Usaría un número real como contador?

   *a*) Sí, porque la máquina los puede representar correctamente.
   *b*) No, porque el ordenador no tiene una representación adecuada para este dato.

8. Si en el algoritmo de la raíz cuadrada se empezara con G = A/2 en lugar de con G = 1 ¿Habría ventajas en el cálculo?

   *a*) Sólo nos ahorraríamos un ciclo de cálculo.
   *b*) Sólo calcularíamos la mitad de ciclos hasta encontrar resultado.

# 6.

# PRUEBAS DE UN PROGRAMA

## CUADRO RESUMEN DE LA LECCION

Prueba de un programa

- Requisitos de las pruebas
- Tipos de errores
- Control de programas por medio de las salidas
- Corrección de programas
  - Corrección de programas que emplean subrutinas
  - Instrucciones que ayudan a la corrección.

Una vez finalizada la codificación, el proceso siguiente es el de la prueba del programa. Su finalidad es comprobar que los resultados sean los esperados. Este proceso desemboca en la puesta a punto del programa.

En este capítulo se desarrollan las características que deben tener esas pruebas, el modo de realizarlas, cómo se puede conocer que un programa da los resultados esperados y qué ayudas se tienen para la corrección del programa.

## 6.1. REQUISITOS QUE DEBEN CUMPLIR LAS PRUEBAS

La metodología que debe seguirse en esta etapa de la resolución de un problema por ordenador se resume en los dos puntos:

1. Utilización del mismo juego de pruebas de ensayo que las usadas en las pruebas de mesa.
2. Las pruebas han de satisfacer todos los resultados esperados.

El programa se ejecuta con el mismo juego de pruebas de mesa, desarrollado en el proceso de diseño. Los resultados dados por la máquina, tienen que ser idénticos a los encontrados cuando se realizaron las pruebas manualmente. En caso de que esto suceda así esta fase de prueba queda concluida y el programa queda listo para la fase siguiente, la documentación, que no introduce ninguna modificación en el programa.

## 6.2. TIPOS DE ERRORES

Los errores que pueden ocurrir en un programa son de dos tipos;

— *Los errores de lógica,* que ocurren en la fase de diseño y que deben ser remediados en esa misma fase mediante el juego de pruebas.

— *Los errores de codificación,* ocurridos durante la fase del mismo nombre y que serán remediados de varias formas. Algunos pueden remediarse en una relectura de la codificación efectuada antes de la prueba del programa, eliminándose algunos de los errores sintácticos y lógicos que estuvieran en el organigrama pero que *no* se hubieran implementado.

El resto de los errores sintácticos se ven en la *fase de prueba* y son puestos de manifiesto cuando el compilador entra en juego para obtener un programa objeto a partir del programa fuente.

Este proceso debe ser efectuado en la fase de diseño. Si se diseñó eficientemente el juego de pruebas, se tendrán resultados satisfactorios a partir de la primera prueba, ahorrando tiempo de ordenador en sucesivas pruebas.

## 6.3. CONTROL DEL PROGRAMA POR MEDIO DE LAS SALIDAS

Los errores que pueden darse a partir de las salidas del programa son:

1. Salida de una información que *no* debe aparecer.
2. Ausencia de una salida que *sí* debe estar.
3. Salida repetida.
4. Salida aleatoria.

Un programa puede dar una salida de una información que no debe aparecer en los resultados. Esto, desde luego, sería motivo de error del programa.

El error consiste en que se ha codificado de modo que se dé salida a un cálculo intermedio o a una información que no pertenece estrictamente al conjunto de los resultados. Se verifica en qué punto del programa sucede y se elimina dicha instrucción de salida, de este modo el error desaparece.

También el programa puede producir una *ausencia de salida de una información que debe aparecer.* Puesto que cada salida proviene de un cálculo y éste está diseñado a partir de una estructura, directamente se busca el problema donde se encuentra esa estructura. Puede ocurrir que no exista o que simplemente no se haya puesto la orden de escritura. Si el programa se ha

diseñado con los criterios de la programación estructurada no será difícil la inclusión de esta nueva estructura.

Otra de las causas de errores es la salida repetitiva de una información que sólo debe aparecer una vez. Este es un error en una estructura repetitiva. El error consiste en haber colocado una instrucción de salida de resultados dentro de un bucle de cálculo, cuando en realidad sólo debía aparecer al final de ese cálculo o al comienzo.

Para solucionar este error se saca esa instrucción de escritura fuera del bucle.

Otra de las causas de error es la salida aleatoria de una información que sólo debe aparecer en determinadas condiciones en el programa. En este caso la búsqueda del error es más laboriosa. Deben encontrarse cuáles son las que dan lugar a que aparezca esa salida. Puesto que una salida es un punto de la estructura, debe investigarse bajo qué condiciones se llega a ese punto del programa, lo que no será difícil si se han seguido las técnicas de la programación estructurada. Deben verificarse también los puntos en que se tienen que hacer los tests a esas condiciones para dejar pasar el flujo del programa hacia las distintas tareas. Una de estas tareas es la de escritura. Esa búsqueda se realizará con la ayuda de la técnica de *parada del programa.*

A continuación se presentan una serie de ejemplos que ponen de manifiesto los errores y los métodos de corrección.

*Ejemplo* 1: *Salida de información que no debe aparecer y Salida aleatoria*

Calcular la intensidad de corriente que pasará por una resistencia de 10 ohmios cuando se le aplica un valor de tensión.

El cálculo de la corriente a partir de la tensión se hace mediante la famosa ley de Ohm que puede expresarse matemáticamente como:

$$\text{Intensidad de corriente} = \frac{\text{Tensión aplicada}}{\text{Valor de la resistencia}}$$

Como condición imponemos que el programa, una vez efectuado un cálculo, siga pidiendo datos de entrada hasta que se encuentre con un dato de entrada que tenga valor nulo, en cuyo caso parará la ejecución.

La descripción del algoritmo es la siguiente (Fig. 6.1, página 126):

Descripción Ø:

EMPIEZA
*Repetir:*
Calcular el valor de la intensidad de corriente.
*Mientras:*
El valor de la tensión no sea cero.
FINAL

Descripción 1:

EMPIEZA
*Repetir:*
Entrar dato de tensión; V.
Calcular el valor de la intensidad; I = V/1Ø.
Escribir V.
*Hasta:* que V sea cero; V = Ø.
FINAL

La codificación de este programa en el lenguaje BASIC es la siguiente:

```
1Ø  READ V
21  I = V/1Ø
31  PRINT V
41  IF (V <> Ø) GOTO 1Ø
51  STOP: END
```

Cuando se va a ejecutar la prueba del programa puede verse que el valor del resultado que sale es el mismo valor que se tiene en la entrada.

Esto es lo que debe salir porque en la codificación se ha escrito equivocadamente la letra V por la I (instrucción 31).

Cambiando en las zonas de escritura I por V, todas las instrucciones permanecen inalteradas salvo la 31 que queda:

```
31  PRINT I
```

La siguiente ejecución del programa demuestra que el programa escribe un cero en último lugar, respondiendo a la entrada de valor cero, encargada de finalizar el proceso, lo cual es erróneo.

El error se corrige colocando el test de V = Ø justo después de la entrada del dato y antes de cualquier cálculo. El organigrama corregido se muestra en la figura 6.2. El programa queda entonces codificado así:

```
10  READ V
21  IF (V = Ø) THEN STOP
31  I = V/1Ø
41  PRINT I
51  GOTO 1Ø
61  END
```

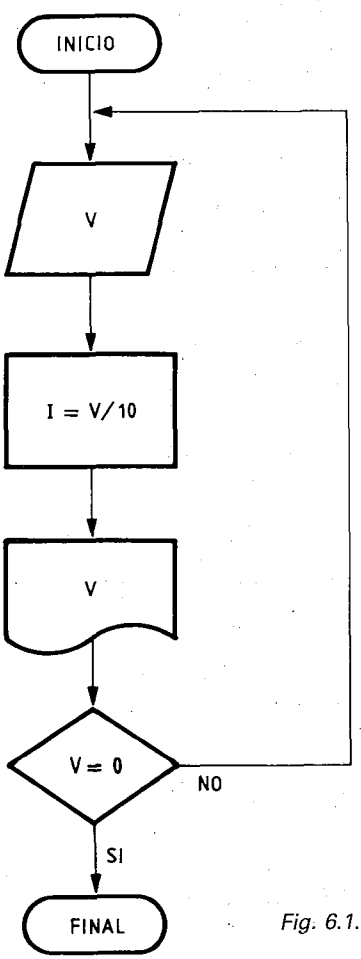



*Fig. 6.1.*

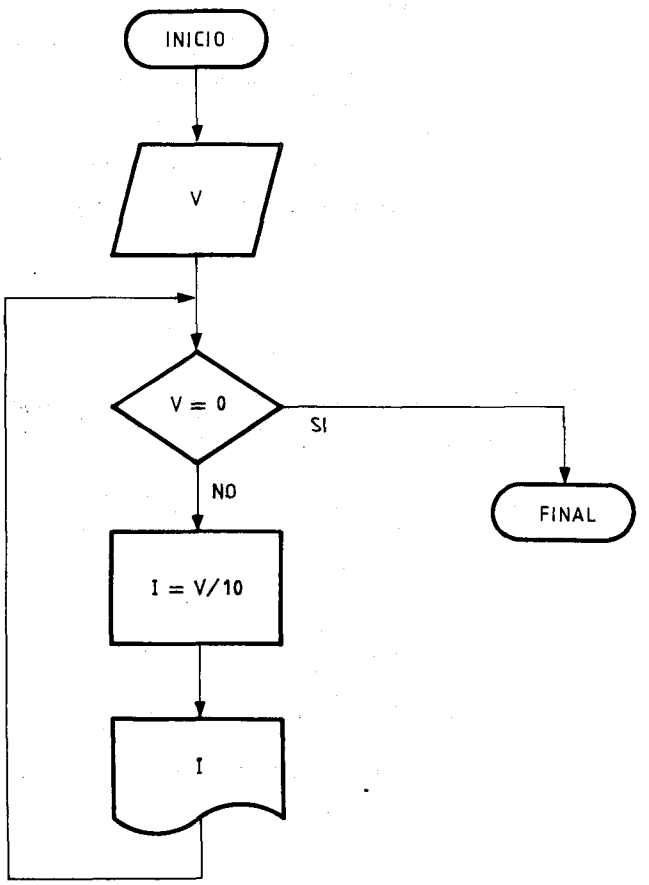



*Fig. 6.2.*

*Ejemplo 2: Salida repetitiva*

Cálculo de la suma de 10 números. La descripción es la siguiente:

Descripción Ø:

EMPIEZA
    Calcular la suma de 10 números.
FINAL

Descripción 1:

EMPIEZA
    *Repetir:*
        Entrar un número.
        Acumular su suma.
    *Hasta:*
        Que hayan entrado los 10 números.
    Dar el resultado de la suma.
FINAL

El organigrama correspondiente a la descripción puede verse en la figura 6.3.

La codificación correspondiente es:

```
1Ø  S =Ø
15  FOR I = 1 TO Ø
21  READ N
41  PRINT S + N
51  NEXT I
61  STOP: END
```

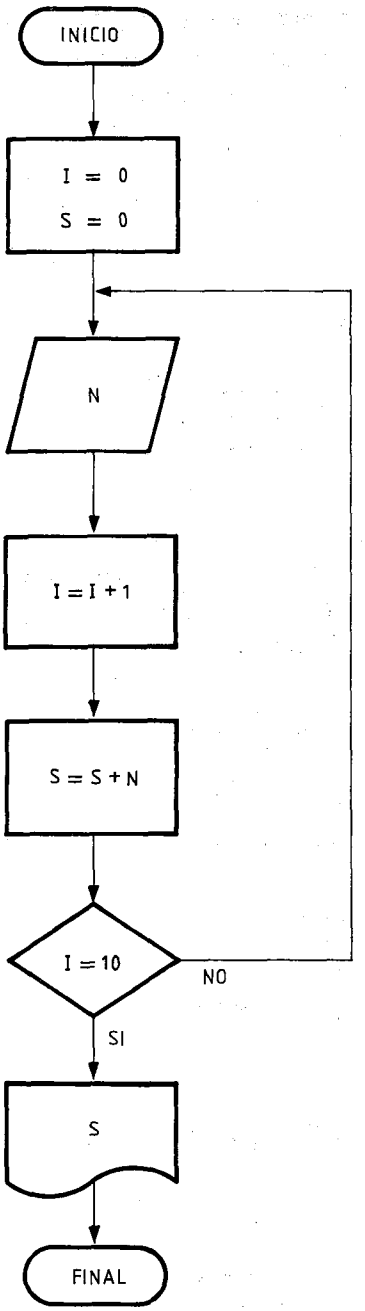



*Fig. 6.3.* ▶

La ejecución del programa así codificado da como salida una serie de 1Ø cifras, de las que sólo la última de ellas es el resultado correcto, mientras que las otras sólo son resultados parciales de dicha suma.

Las descripciones y el organigrama son correctos: primero efectuamos el cálculo y, fuera del bucle de cálculo, está la salida. Entonces debe deducirse que el error se encuentra en la codificación. En efecto, se han unido en una instrucción la acumulación de la suma y la escritura del resultado, es decir, S = S + N y PRINT S se han escrito conjuntamente como PRINT S + N (instrucción 41).

Esto produce, puesto que está dentro del bucle, que se vayan escribiendo todos los resultados parciales. La solución del error está pues en la separación de esas dos instrucciones, colocando la escritura, después de haber efectuado el cálculo con la totalidad de los diez números, fuera del bucle.

Las modificaciones en el programa dan:

```
1Ø  S = Ø
15  FOR I = 1 TO 1Ø
21  READ N
41  S = S + N
51  NEXT I
61  PRINT S
71  STOP: END
```

## 6.4. CORRECCION DE PROGRAMAS

Se comentan las técnicas y las herramientas que tienen los programadores para la fase de corrección.

### 6.4.1. Corrección de programas que emplean subrutinas

Puesto que la subrutina es realmente un programa completo, empieza y acaba en lugares determinados, las subrutinas se prueban separadamente.

Una vez que el programador se ha cerciorado del correcto funcionamiento de la subrutina, pueden enlazarse todas ellas para terminar el programa.

### 6.4.2. Instrucciones que ayudan a la corrección

En algunos sistemas de ordenadores las instrucciones introducidas para la corrección pueden ir marcadas de alguna manera (un color diferente de la ficha, una clave en la ficha, etc.) que permite, una vez concluida con éxito la fase de corrección, eliminar estas instrucciones.

— Instrucciones de escritura.
— Instrucciones de parada.

#### *Instrucciones de escritura*

Para conocer qué es lo que ocurre en el interior del ordenador, éste debe escribir el estado de sus variables, mediante las instrucciones de escritura.

Para ver cómo se va desarrollando un cálculo, pueden colocarse instrucciones de escritura para las variables intermedias de ese cálculo y se va viendo cómo varían de valor hasta llegar a la solución final.

En las pruebas de mesa se había considerado oportuno la construcción de una tabla del estado interno de cada una de las variables del ordenador. Puesto que se posee esa tabla, se pueden comparar los resultados ahí escritos con los que proporciona la máquina.

Una comprobación importante consiste en ver si el programa lee correctamente los datos que se le introducen. Es pues conveniente colocar instrucciones de escritura inmediatamente después de las instrucciones de lectura, para asegurarse de cuál es el dato que ha tomado como entrada.

Las instrucciones deben llevar algún comentario para distinguir a qué variable pertenecen. De otro modo no se sabría dónde corresponde cada uno de los números que aparecen en la pantalla o escritos en el papel.

Pueden ser instrucciones de este tipo:

```
1Ø PRINT «A = »;A.
```

Con lo que se sabrá que el valor que aparece escrito es el valor que, en aquel instante, corresponde a la variable A.

De esta forma puede conocerse el estado interno de la máquina y el desarrollo de los cálculos.

Para conocer por dónde pasa el flujo del programa también es conveniente colocar instrucciones de escritura. Teniendo el programa estructurado, a cada

una de las ramas o tareas puede dársele un número o un nombre y colocar una instrucción de escritura encabezando esa tarea, de modo que se escriba ese número o nombre cada vez que el flujo pasa por ahí.

Este sistema es conveniente para comprobar si las condiciones se cumplen. Se conoce el valor de las variables y también el camino que ha tomado el flujo en el programa. Es cuestión de comparar únicamente si el camino que ha llevado el programa es el correcto con esos valores de las variables.

### *Instrucciones de parada*

Si se sigue el procedimiento anterior con un programa entero es posible que después de la ejecución se tenga una cantidad tal de valores de variables, de lugares de paso, etc. que resulte costoso su análisis.

Puesto que estas partes son independientes, se puede colocar al final y al principio de ellas instrucciones de parada de la ejecución. Se introducen entonces los valores de las variables que interese comprobar (mediante instrucciones de asignación) y dejamos ejecutarse esa estructura, que poseerá las convenientes instrucciones de escritura para el control del proceso. Al finalizar ésta, el programa encuentra otra instrucción de parada de ejecución y se interrumpe. Tenemos entonces tiempo de estudiar los pocos resultados parciales obtenidos de la ejecución de esa parte del programa. A continuación y una vez decidido que la estructura funciona correctamente se colocan los nuevos valores para la siguiente porción de programas y se da la orden de continuación de la ejecución.

*Ejemplo*

En el ejemplo se ve el proceso de corrección de un programa. El enunciado del ejemplo es: Se desea conocer el mayor de una terna de números.

Los números son A, B y C. Se colocará el máximo en una posición de memoria que llamaremos M. La descripción es la siguiente:

Descripción Ø:

```
EMPIEZA
    Calcular el mayor de tres números: A, B, C.
FINAL
```

Descripción 1:

EMPIEZA
Comparar A con B.
*Si* A es mayor o igual que B *entonces* guardar A en M.
*En caso contrario:* guardar B.
Comparar M con C.
*Si* M es mayor o igual que C *entonces* guardar M.
En caso contrario guardar C.
FINAL

El organigrama correspondiente se muestra en la figura 6.4, (Pág. 132).

Se ha numerado cada uno de los símbolos empleados en el organigrama, para la posterior identificación y comentario.

Bloque 1: entrada de datos
Bloque 2: decisión entre A y B
3: operación en caso afirmativo de la condición.
4: operación en caso negativo de la condición.
5: comparación del mayor entre A y B y del C.
6: operación en caso afirmativo.
7: operación en caso negativo.
8: escritura del resultado.

Las instrucciones que se añaden para la corrección y prueba del programa son las siguientes:

Bloque 2: escritura de los datos leídos.
PARO (instrucción de parada en la ejecución).

Al efectuarse esa interrupción se colocan los valores a que interesen A y B, para comprobar la decisión expresada en el bloque 2».

Bloque 3: escritura de «Bloque 3» para verificar si pasa por ahí.
PARO.

Bloque 4: escritura de «Bloque 4» para verificar el paso por esa alternativa.
PARO.

Puesto que debe haber tomado alguna de esas alternativas en cualquier caso el programa volverá a pararse, momento en que se comentará la verificación de la condición expresada en el bloque 5.

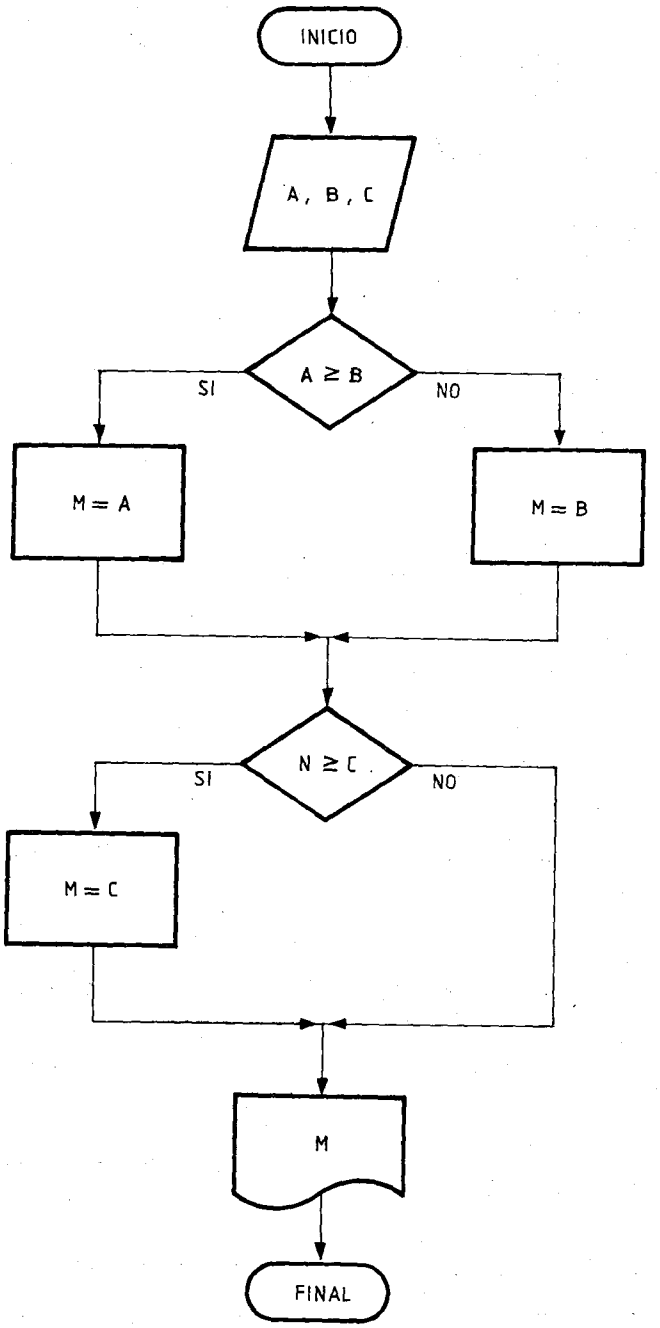



*Fig. 6.4.*

Los valores de las variables deben corregirse si es necesario y darse la orden de continuación. Pero antes debe escribirse:

Bloque 5: escritura del valor de M.

Bloque 7: escritura de «Bloque 7», con el fin de identificación. PARO.

Bloque 8: escritura del valor de M.

Una codificación del problema es la siguiente:

```
1Ø  INPUT A, B, C
2Ø  IF (A B) GOTO 4Ø
23  M = B
3Ø  GOTO 5Ø
4Ø  M = A
5Ø  IF (M = C) THEN 7Ø
6Ø  M = C
7Ø  PRINT M
8Ø  STOP: END
```

Los controles que se intercalan son:

```
15  PRINT "A = ";A;"B = ";B;"C = ";C
18  STOP
24  PRINT "M = ";M,:PRINT "BLOQUE 4": PAUSE
42  PRINT "BLOQUE 2",:PRINT "M = ";M: PAUSE
62  PRINT "M = ";M,:PRINT "BLOQUE 7": PAUSE
63  GOTO 7Ø
```

como producción del programa se tiene:

Datos 1, 2, 3

```
RUN
? 1,2,3
A * + 1 B = 2 C = 3
M = 2 BLOQUE 4
M = 3 BLOQUE 7
3
```

Datos: 3, 2, 1

```
RUN
? 3,2,1
A = 3   B = 2   C = 1
BLOQUE 2
M = 3
3
```

Datos 1, 3, 2

```
RUN
? 1,3,2
A = 1  B = 3  C = 2
M = 3  BLOQUE 4
3
```

## EJERCICIOS

1. Un error de codificación precisa del ordenador para su apreciación

   *a)* No, un repaso manual de la sintaxis nos permite averiguarlo.
   *b)* En todos los casos el compilador tiene que ponerlo de manifiesto.

2. Cuando encontramos que un programa siempre escribe el encabezamiento cada vez que tiene que ejecutar un cálculo nuevo

   *a)* Estamos en presencia de un error en una estructura repetitiva.
   *b)* Estamos en presencia de un error en una estructura alternativa.

3. Las instrucciones de paro del programa

   *a)* Permiten una corrección más rápida y completa de un programa porque se tiene la posibilidad de cambiar los valores de las variables que se manejan.
   *b)* No sirven, porque si lo interrumpimos tarda más tiempo en ejecutarse.

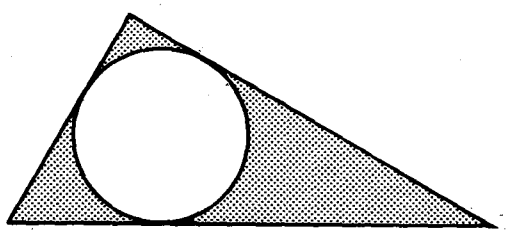
# 7.

# EJEMPLOS DE PROGRAMAS

## 1. CALCULO DE AREAS

Conociendo los tres lados de un triángulo, hallar cuál es el área restante en dicho triángulo al inscribirle una circunferencia tangente a sus tres lados.

La representación gráfica es la siguiente:



Este problema tiene varios pasos:

1. Hallar el área de triángulo.
2. Hallar el radio de la circunferencia.
3. Hallar el área de dicha circunferencia.
4. Restar del área del triángulo el área del círculo inscrito.
5. Ensamblar los distintos pasos efectuados.

— *Area del triángulo:*

El área de cualquier triángulo se determina con la fórmula

$$\text{Area} = \sqrt{S\,(S-a)\,(S-b)\,(S-c)}$$

$$S = \frac{a + b + c}{2} \text{ (semiperímetro)}$$

Por cada una de estas fórmulas se definirá una función así:

```
DEF FNA (A,B,C) = (A+B+C)/2
```

```
DEF FNA (A,B,C) = (A+B+C)/2
DEF FNB (A,B,C) = SQR (FNA (A,B,C)) * (FNA (A,B,C)-A) *
(FNA (A,B,C)-B) * (FNA (A,B,C)-C)
```

— *Radio de la circunferencia*

El radio máximo de una circunferencia inscrita en un triángulo tangente a sus lados, se calcula mediante la fórmula:

$$r = \sqrt{\frac{S\,(S-a)\,(S-b)\,(S-c)}{2}}$$

Como el dividendo de la fórmula es la misma expresión que el área del triángulo, se codificará:

```
DEF FNC (A,B,C) = FNB (A,B,C)/2
```

— *Area de la circunferencia*

El área de la circunferencia es:

$$\text{Area} = \pi\, r^2$$

La codificación de la función que halla el área llamará a su vez a la función que determina cuál debe ser su radio:

```
DEF FND (A,B,C) = &PI * FNC (A,B,C) ↑ 2
```

— *Resta de áreas*

Para restar las áreas del triángulo y del círculo inscrito en él, se restará las funciones que calculan dichas áreas:

```
DEF FNE (A,B,C) = FNB (A,B,C) - FND (A,B,C)
```

## Programa

La codificación del programa sería:

```
10  DEF FNA (A,B,C) = (A+B+C)/2
20  DEF FNB (D,E,F) = SQR (FNA (D,E,F)) * (FNA (D,E,F)-
    D)* (FNA (D,E,F)-E) * (FNA (D,E,F)-F)
30  DEF FNC (G,H,I) = FNB (G,H,I)/2
40  DEF FND (J,K,L) = &PI * FNC (J,K,L) ↑ 2
50  DEF FNE (M,N,O) = FNB (M,N,O) - FND (M,N,O)
.
.
.
10550  A = FNE (X1, X2, X3)
10560  ————————
```

Debe recordarse que las variables de las definiciones de estas funciones son argumentos mudos y, por tanto, no tienen nada que ver con las variables A, B, C, D, E, F, G, H, I, J, K, L, M, N y O que contenga el programa.

En la instrucción 10550, se asigna a la variable numérica A el valor de la función numérica FNE, la cual utiliza las variables numéricas X1, X2 y X3 que almacenan el valor de los tres lados del triángulo del problema que se acaba de desarrollar, y que se explicará a continuación cómo funciona:

— La instrucción 10550, llama a la función FNE asignando a los argumentos mudos de su definición M, N y O los valores de las variables X1, X2 y X3 respectivamente.

— La función FNE llama en primer lugar a la función FNB para calcular el área del triángulo, ésta llama a su vez a la función FNA para que le retorne el valor del semiperímetro que necesita para calcular dicha área.

— Una vez obtenida el área del triángulo, la función FNE llama a la función FND para calcular el área del círculo inscrito, la cual llama a la función FNC para obtener el radio de dicho círculo, y FNC llama a su vez a FNB (y esta a FNA) para realizar dicho cálculo.

— Una vez se han realizado todos los cálculos, el valor de la función FNE será la resta de los valores de la función FNB y de la función FND, valor es almacenado en la variable numérica A.

— El procesador pasa a ejecutar la instrucción 10560.

## 2. MEDIA ARITMETICA Y MEDIA GEOMETRICA

Calcular la media aritmética y la media geométrica de los n primeros números naturales.

*Datos necesarios:*

Cantidad de números (n).

*Resultado:*

Media aritmética y media geométrica.

*Fórmulas:*

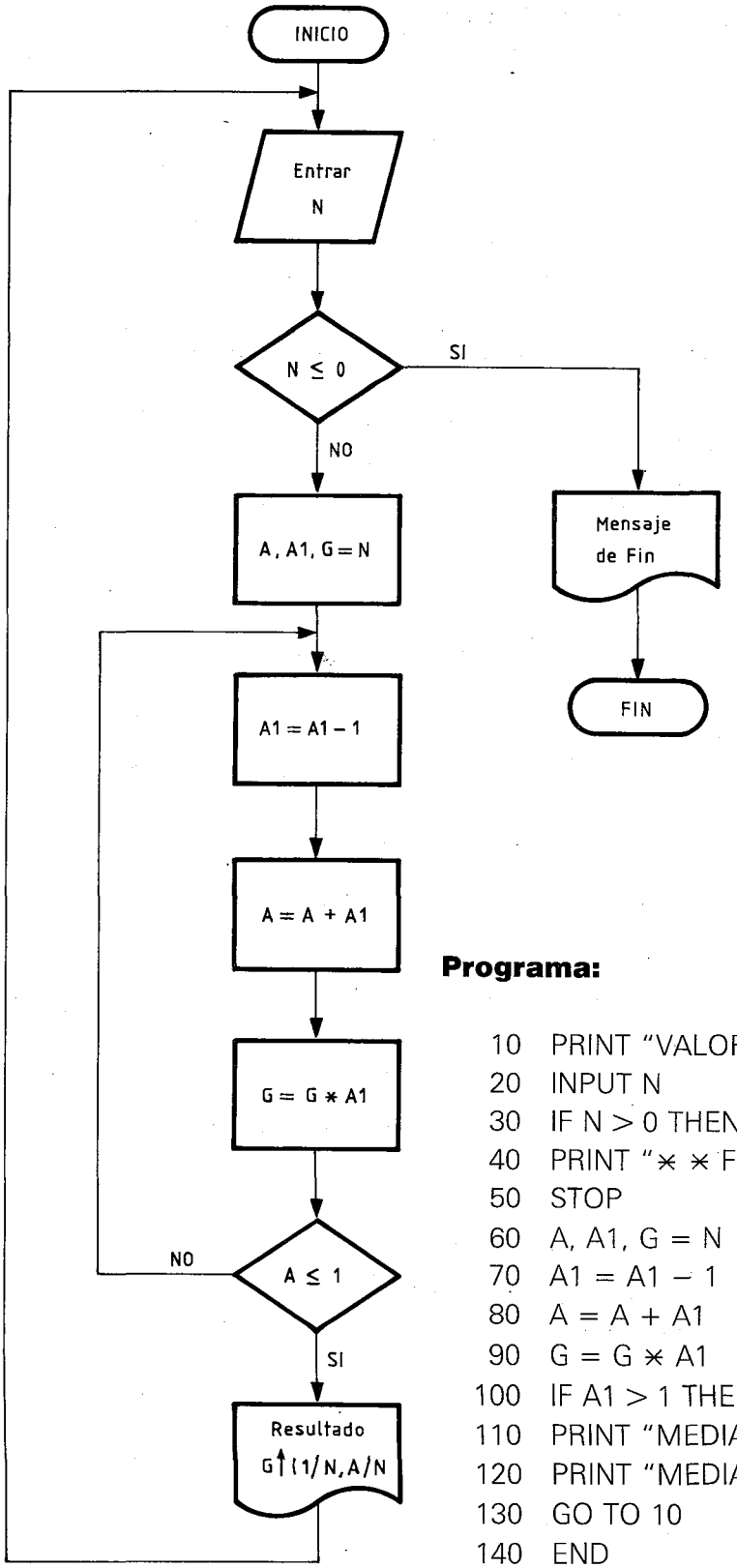
$$MA = \frac{\sum_{i=1}^{n} Ni}{n}$$

$$MG = (N_1 * N_2 * \cdots * N_n)^{1/n}$$

siendo:

Ni = Un número natural.
n = Cantidad de números naturales.
$\sum_{i=1}^{n}$ = Sumatorio (desde i = 1 hasta n).
i = Número natural.
MA = Media aritmética.
MG = Media geométrica.

## ◄ Organigrama




**Programa:**

```
 10  PRINT "VALOR DE N";
 20  INPUT N
 30  IF N > 0 THEN 60
 40  PRINT "* * FIN * *"
 50  STOP
 60  A, A1, G = N
 70  A1 = A1 – 1
 80  A = A + A1
 90  G = G * A1
100  IF A1 > 1 THEN 70
110  PRINT "MEDIA ARITMETICA="; (A/N),
120  PRINT "MEDIA GEOMETRICA="; [G ↑ (1/N)]
130  GO TO 10
140  END
```

## 3. PROBLEMA DE BALISTICA

Calcular el alcance máximo y el tiempo de alcance de una granada disparada por un mortero.

*Datos necesarios:*

— Angulo de disparo (grados)

— Velocidad con que la granada sale del mortero (metros/sg.)

*Resultado:*

— Alcance en metros.

— Tiempo en segundos.

*Fórmulas:*

$$X = \frac{V_0^2 * \text{sen } 2\varphi}{g}$$

$$T = \frac{2xV_0 * \text{sen } \varphi}{g}$$

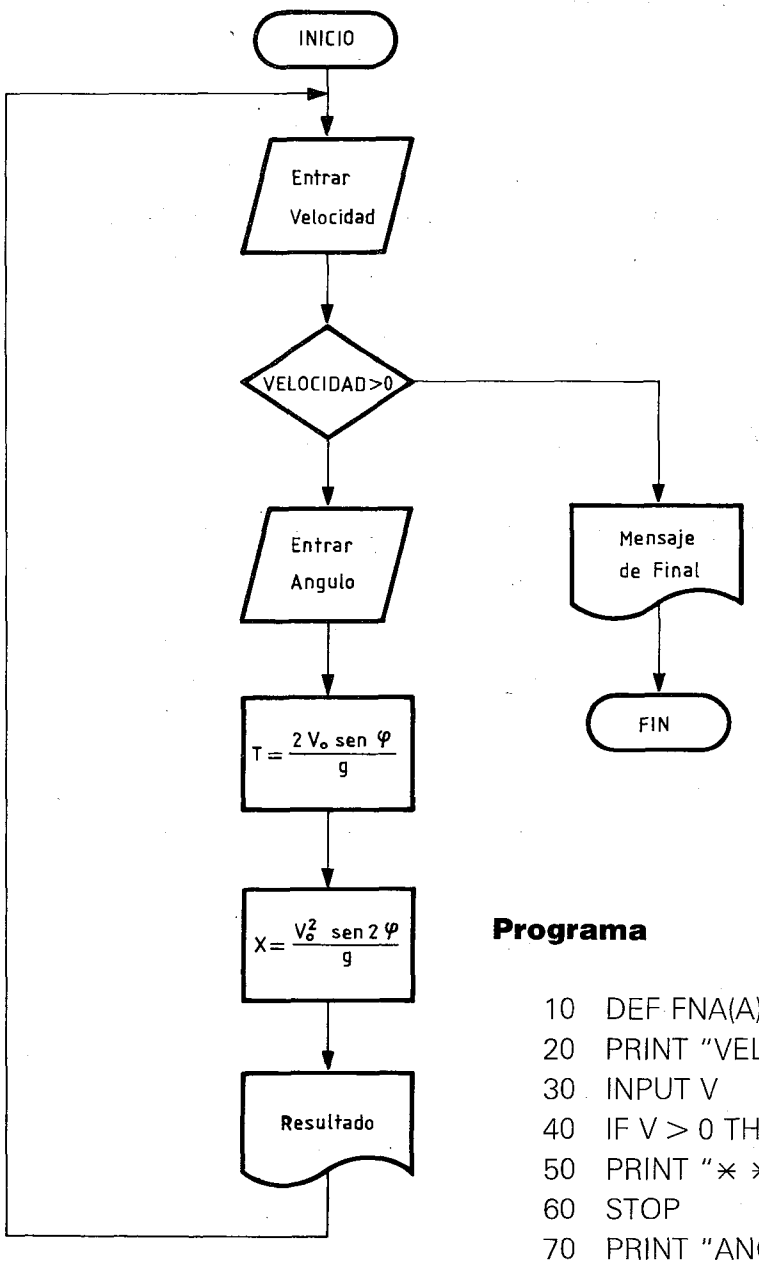
siendo:

$X$ = alcance.
$V_0$ = Velocidad inicial.
$\varphi$ = ángulo de disparo.
$g$ = aceleración de la gravedad (=9.8).
$T$ = Tiempo de alcance.

*Fin del proceso:*

Velocidad inicial = 0.

## ◄ Organigrama




## Programa

```
 10  DEF FNA(A) = SIN[RAD(A)]
 20  PRINT "VELOCIDAD INICIAL";
 30  INPUT V
 40  IF V > 0 THEN 70
 50  PRINT "* * FIN * *"
 60  STOP
 70  PRINT "ANGULO";
 80  INPUT A
 90  T = 2 * V * FNA(A)/9.8
100  X = V * * 2 * FNA(A *)/9.8
110  PRINT "ALCANCE ="; X
120  PRINT "TIEMPO ="; T
130  PRINT
140  GO TO 20
150  END
```

## 4. PROBLEMA SIMULADO DE BALISTICA

Derribar un avión antes de que bombardee la batería antiaérea.

*Datos necesarios:*

— Batería antiaérea:

  — ángulo mínimo de disparo (grados)

  — ángulo máximo de disparo (grados)

  — precisión angular (grados)

  — velocidad del obús (metros/sg.)

  — tiempo mínimo para preparar el disparo (segundos)

— Avión:

  — distancia a la que se encuentra (metros)

  — velocidad a la que se aproxima

  — altura (metros/sg)

*Resultados:*

— Angulo de disparo (grados)

— Tiempo de espera hasta abrir fuego (segundos)

*Fórmulas:*

— Tiempo que tarda el avión desde la posición en que ha sido detectado hasta alcanzar la posición para bombardear la batería antiaérea

$$T_1 = \frac{X_1}{V_1} - \sqrt{\frac{2}{g} xy_1}$$

siendo:

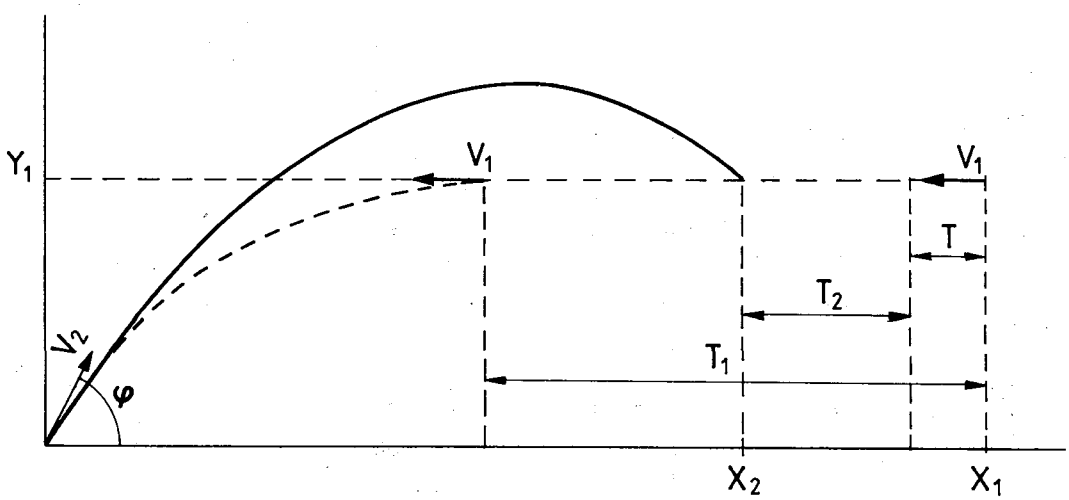
$X_1$ = posición en que ha sido detectado el avión.
$V_1$ = velocidad del avión.
$Y_1$ = altura a la que vuela el avión.
g = aceleración de la gravedad (= 9.8) (2/9.8 = 0.20408163)

— Distancia del avión a la que hay que abrir fuego en función del ángulo de disparo de la batería antiaérea, de la velocidad del obús y de la altura a la que se encuentra el avión.

$$x = \cos \varphi * \frac{V_2 * \text{sen } \varphi + \sqrt{V_2^2 * \text{sen } \varphi - 2 * g \times Y_1}}{g}$$

siendo:

x = distancia del avión
$\varphi$ = ángulo de disparo
$V_2$ = velocidad del obús
g = aceleración de la gravedad (= 9.8) (2 * 9.8 = 19.6)
$Y_1$ = altura del avión.



— Tiempo de espera para abrir fuego desde que se ha detectado el avión hasta que alcanza la distancia máxima en la que es alcanzado por el fuego de la batería.

$$T_2 = \frac{X_1}{V_1} - \frac{X_2}{V_2} - T$$

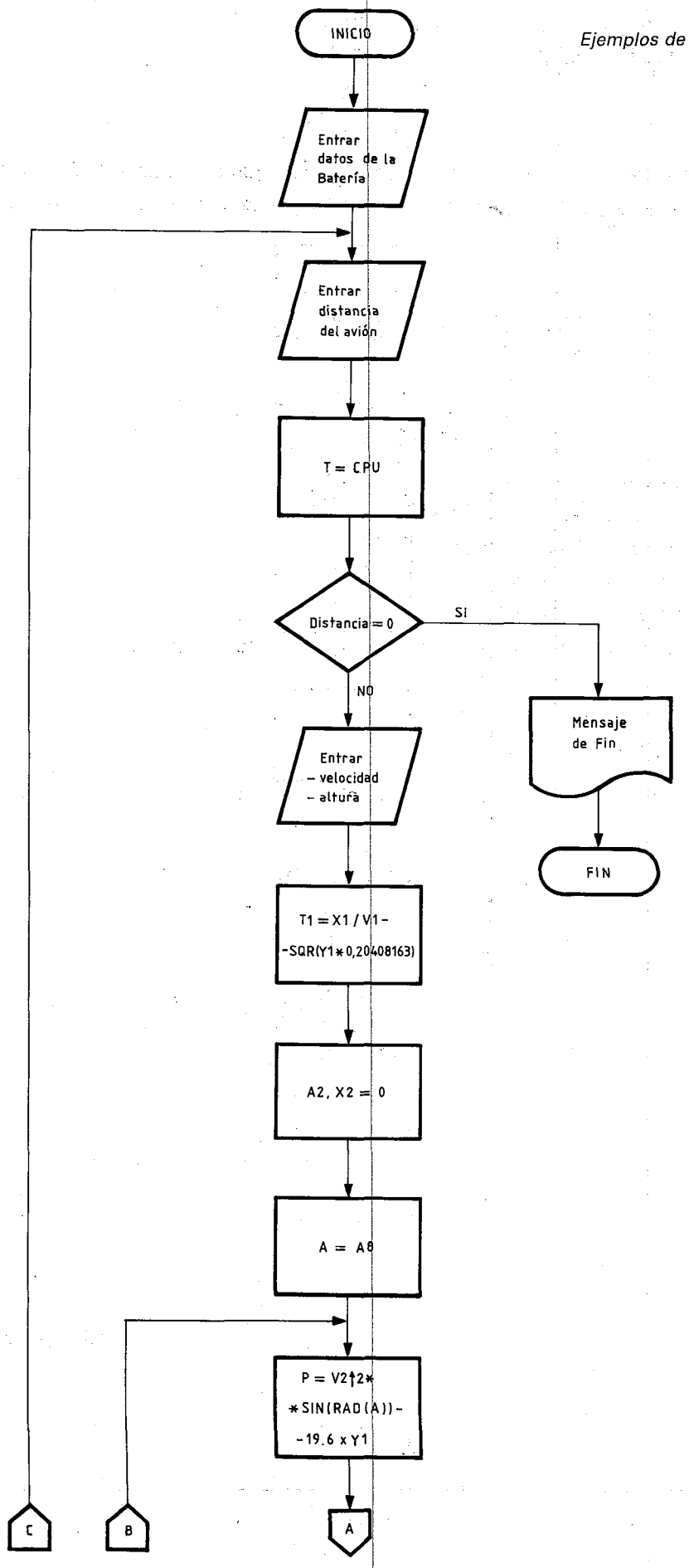
siendo:

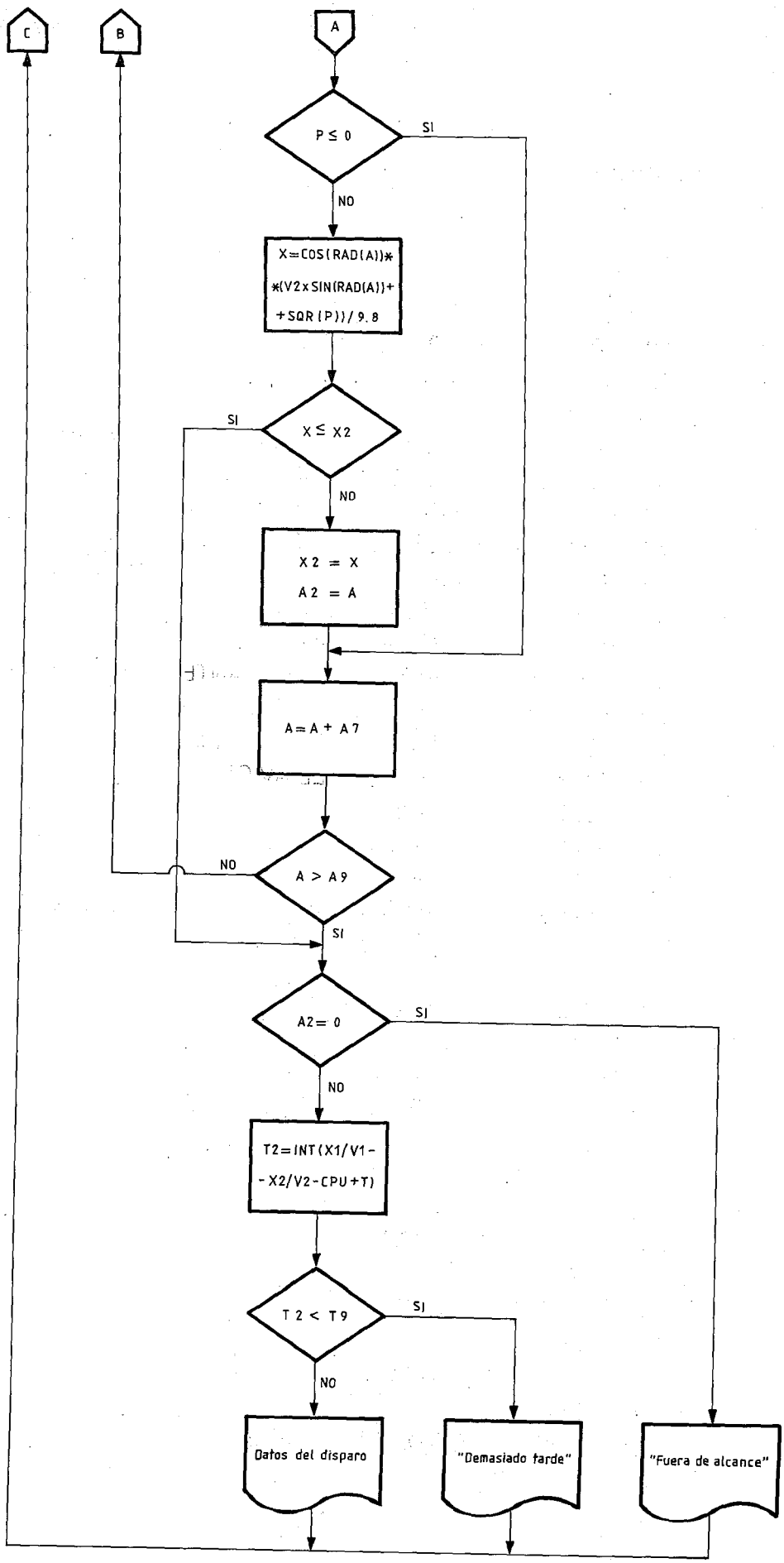
$T_2$ = tiempo de espera
$X_1$ = distancia en que se ha detectado el avión
$X_2$ = distancia máxima para abrir fuego
$V_1$ = velocidad del avión
$V_2$ = velocidad del obús
$T$ = tiempo que ha tardado el ordenador en sus cálculos

*Fin del proceso:*

distancia en la que se ha detectado el avión = 0

**Organigrama: ▶**


INICIO
Entrar datos de la Batería
Entrar distancia del avión
T = CPU
Distancia = 0
SI
NO
Mensaje de Fin
FIN
Entrar
– velocidad
– altura
T1 = X1 / V1 –
–SQR(Y1 * 0,20408163)
A2, X2 = 0
A = A8
P = V2↑2 *
* SIN(RAD(A)) –
–19.6 x Y1
C
B
A


C
B
A
P ≤ 0
SI
NO
X=COS(RAD(A))*
*(V2xSIN(RAD(A))+
+SQR(P))/9.8
X ≤ X2
SI
NO
X 2 = X
A 2 = A
A=A + A7
A > A9
NO
SI
A2= 0
SI
NO
T2=INT(X1/V1-
-X2/V2-CPU+T)
T 2 < T9
SI
NO
Datos del disparo
"Demasiado tarde"
"Fuera de alcance"

**Programa:**

```
 10 REM DERRIBO DE AVIONES
 20 REM ————————
 30 REM
 40 REM A = ANGULO DE MUESTREO DEL DISPARO
 50 REM A2 = ANGULO DE DISPARO
 60 REM A7 = PRECISION DE LA BATERIA
 70 REM A8 = ANGULO DE DISPARO MINIMO DE LA BATERIA
 80 REM A9 = ANGULO DE DISPARO MAXIMO DE LA BATERIA
 90 REM P = TECHO DEL DISPARO
100 REM T = TIEMPO DEL CALCULO
110 REM T1 = TIEMPO DISPONIBLE ANTES DEL BOMBARDEO
120 REM T2 = TIEMPO PARA ABRIR FUEGO
130 REM T9 = TIEMPO NECESARIO PARA DISPARAR
140 REM V1 = VELOCIDAD DEL AVION
150 REM V2 = VELOCIDAD DEL OBUS
160 REM X = ALCANCE SEGUN TECHO
170 REM X1 = DISTANCIA DEL AVION
180 REM X2 = DISTANCIA ABRIR FUEGO CONTRA EL AVION
190 REM Y1 = ALTURA DEL AVION
200 REM
210 REM *DATOS DE LA BATERIA*
220 REM
230 PRINT "————————"
240 PRINT "CARACTERISTICA DE LA BATERIA"
250 PRINT "————————"
260 PRINT "ANGULO MINIMO";
270 INPUT 48
280 PRINT "ANGULO MAXIMO";
290 INPUT A9
300 PRINT "PRECISION";
310 INPUT A7
320 PRINT "TIEMPO MINIMO";
330 INPUT T9
340 PRINT "VELOCIDAD DEL OBUS";
350 INPUT V2
```

```
360  PRINT "————————————"
370  REM
380  REM *DATOS DEL AVION*
390  REM
400  PRINT "DISTANCIA";
410  INPUT X1
420  T = CPU
430  IF X1 = 0 THEN 460
440  PRINT "** FIN **"
450  STOP
460  PRINT "VELOCIDAD";
470  INPUT V1
480  PRINT "ALTURA";
490  INPUT Y1
500  REM
510  REM *FALTA T1 PARA EL BOMBARDEO*
520  REM
530  T1 = X1/V1-SQR(Y1*0.20408163)
540  REM
550  REM *ANGULO DE DISPARO*
560  REM
570  A2,X2=0
580  FOR A=A8 TO A9 STEP A7
590  P=V2 ↑ 2*SIN(RAD(A))-19.6* y 1
600  IF P ≤ 0 THEN 650
610  X=COS(RAD(A))*(V2*SIN(RAD(A))+SQR(P))/9.8
620  IF X ≤ X2 THEN 660
630  X2=X
640  A2=A
650  NEXT A
660  IF A2=0 THEN 790
670  REM
680  REM *ABRIR FUEGO DENTRO DE T2*
690  REM
700  T2=INT(X1/V1-X2/V2-CPU+T)
710  IF T2 < T9 THEN 810
720  REM
730  REM *DATOS DEL DISPARO*
740  REM
750  PRINT "ANGULO=";A2
```

```
760  PRINT "ABRIR FUEGO EN";T2
770  PRINT "————————————"
780  GO TO 400
790  PRINT "*FUERA DE ALCANCE*"
800  GO TO 770
810  PRINT "*DEMASIADO TARDE*"
820  GO TO 770
830  END
```

## SOLUCIONES DE LOS EJERCICIOS

### 2. Solución de un problema por ordenador

1.– *b*); 2.– *a*); 3.– *b*); 4.– *a*) 5.– *b*); 6.– *b*); 7.– *a*); 8.– ; 9.– *c*); 10.– *c*); 11.– *b*); 12.– *b*); 13.– *c*); 14.– *c*).

### 3. Construcción de algoritmos y programas

1.– *a*); 2.– *a*); 3.– *b*); 4.– *a*); 5.– *c*); 6.– *c*); 7.– *b*); 8.– *a*); 9.– *c*); 10.– *b*); 11.–*b*); 12.– *c*); 13.– *a*); 14.– *a*); 15.– *a*); 16.– *c*); 17.– *a*); 18.– *a*).

### 4. Diseño de un programa.

1.– *b*); 2.–*a*); 3.– *c*); 4.– *a*); 5.– *a*); 6.– *a*); 7.– *b*); 8.– *c*); 9.– *c*).

### 5. Codificación

1.– *a*); 2.– *b*); 3.– *b*); 4.– *b*); 5.– *b*); 6.– *c*); 7.– *b*); 8.– *a*).

### 6. Pruebas de un programa

1.– *a*); 2.– *a*); 3.– *a*).

# Indice General

Contenido ..... 7

Introducción ..... 9

## 1. UTILIZACION DEL ORDENADOR

Cuadro resumen ..... 15
1.1. Trabajos de creación ..... 16
1.2. Ayudas a la educación ..... 17
1.3. Gestión de datos y preparación de informes comerciales ..... 18
1.4. Ayuda a la administración del Estado ..... 18
1.5. Cálculo científico y técnico ..... 19
1.6. Control de procesos ..... 20
1.7. Inteligencias artificiales ..... 20

## 2. FASES DE LA PROGRAMACION

Cuadro resumen ..... 25
2.1. Identificación del problema y definición de objetivos ..... 26
2.2. Descripción matemática ..... 27
2.3. Análisis numérico ..... 27
2.4. Programación ..... 27
2.5. Prueba del programa ..... 34
2.6. Documentación ..... 36

2.7. Producción del programa ..... 37
2.8. Interpretación de los resultados ..... 37

Ejercicios ..... 38

## 3. CONSTRUCCION DE ALGORITMOS Y PROGRAMAS

Cuadro resumen ..... 43
3.1. Reglas en la confección de algoritmos ..... 44
3.2. Algoritmos de exploración de datos ..... 45
3.2.1. Exploración secuencial ..... 45
3.2.2. Algoritmo de exploración binaria ..... 47
3.2.3 Algoritmo binario con interpolación ..... 48
3.2.4. Exploración en árbol binario ..... 51
3.2.5. Algoritmo de condensación ..... 53
3.3. Características de un programa ..... 53
3.4. Criterios en la construcción de programas ..... 55
3.4.1. Criterio de normalización ..... 55
3.4.2. Criterio de realización ..... 56

Ejercicios ..... 60

## 4. DISEÑO DE UN PROGRAMA

Cuadro resumen ..... 65
4.1. Programación modular ..... 66
4.2. Programación estructurada ..... 69
4.2.1. Metodología arriba-abajo ..... 72
4.2.2. Uso de recursos abstractos ..... 73
4.2.3. Estructuras de la información ..... 73
— Estructura secuencial ..... 73
— Estructura repetitiva ..... 74
— Estructura alternativa ..... 78
4.3. Pruebas de mesa ..... 86

Ejercicios ..... 91

## 5. CODIFICACION

Cuadro resumen ..... 95
5.1. Subprogramas ..... 96

5.2. Codificación de las estructuras ..... 106
5.2.1. Codificación de la estructura secuencial ..... 106
5.2.2. Codificación de la estructura alternativa ..... 107
5.2.3. Codificación de las estructuras repetitivas ..... 107
5.3. Estructuras de datos ..... 108
5.4. Tipos de datos ..... 109

Ejercicios ..... 117

**6. PRUEBAS DE UN PROGRAMA**

Cuadro Resumen ..... 121
6.1. Requisitos que deben cumplir las pruebas ..... 122
6.2. Tipos de errores ..... 123
6.3. Control del programa por medio de las salidas ..... 123
6.4. Corrección de programas ..... 128
6.4.1. Corrección de programas que emplean subrutinas ..... 128
6.4.2. Instrucciones que ayudan a la corrección ..... 129
— Instrucciones de parada ..... 130
— Instrucciones de escritura ..... 129

Ejercicios ..... 134

**7. EJEMPLOS DE PROGRAMAS**

1. Cálculo de áreas ..... 136
2. Media aritmética y media geométrica ..... 140
3. Problema de balística ..... 142
4. Problema simulado de balística ..... 144

Soluciones de los ejercicios ..... 153

El tomo de Leyes de Construcción de Programas define reglas para la construcción de procedimientos para las operaciones en un ordenador.

El ordenador es simplemente una herramienta que impone las posibilidades de su tecnología. La finalidad de esta obra consiste en definir las leyes de utilización que permiten obtener el máximo rendimiento del mismo. Para ello se estudia el modo en que deben tratarse los datos.

En definitiva, esta obra presenta un conjunto de leyes para la organización de las secuencias de instrucciones (algoritmos) así como para la organización de los datos que constituyen la base del tratamiento de la información.

*El dibujo de la cubierta es cortesía de* GENERAL DE COMPUTADORES, S.A.