BASIC

Introducción a la programación

J. C. LARRÉCHÉ

LeG PARANINFO SA

BASIC

Introducción a la programación

Jean-Claude LARRÉCHÉ

BASIC

Introducción a la programación

CUARTA EDICION

1984

PARANINFO S.A.

MADRID

Traducido por:
F. J. SANCHIS LLORCA
Dr. Ingeniero Industrial. Licenciado en Informática.
Senior T.S. de IBM, S.A.E. Profesor de la Facultad de
Informática de la Universidad Politécnica de Madrid

Esta obra es la traducción del libro francés de J.C. LARRECHE:
"LE BASIC. UNE INTRODUCTION A LA PROGRAMMATION"

IMPRESO EN ESPAÑA
PRINTED IN SPAIN

ISBN: 84-283-1215-X

Depósito Legal: M-26201-1984

PARANINFO S.A. Magallanes, 25 - MADRID (15) (5-3399)

ALCO, artes gráficas. Jaspe, 34 - Madrid-26.

Indice de materias

Prólogo

El ordenador ya no es una herramienta reservada a los especialistas. Un gran número de investigadores, de funcionarios y de estudiantes saben cómo funciona un ordenador y lo que éste puede o no hacer. Estos últimos años han aparecido varios libros de introducción a la Informática, en los que se explica de forma sencilla la construcción y el funcionamiento de un ordenador.

Los ejecutivos que trabajan en las universidades, ministerios o empresas de un cierto tamaño tienen hoy la posibilidad de acceder a un ordenador. Sin embargo, son raros los que usan personalmente un ordenador como herramienta que les permite mejorar sus decisiones. Cuando un estudio es demasiado largo y complejo para ser realizado manualmente, entonces se encarga a un especialista o sencillamente se ignora. A menudo este estudio puede resolverse muy simplemente empleando un ordenador y no es interesante para un especialista que prefiere los problemas más complejos y que, de todas formas, está desbordado de trabajo. También con frecuencia, el estudio mencionado, al mejorar la calidad de una decisión, se traduciría en costes de proceso reducidos y en beneficios netos para la empresa (sin tener en cuenta que el ordenador a menudo está infrautilizado y que los costes marginales son despreciables).

¿Por qué los ejecutivos que tienen acceso a un ordenador y conocen sus posibilidades no los emplean más para mejorar la eficacia de la empresa? A mi parecer, la dificultad principal es la siguiente: para resolver un problema con ordenador, es necesario escribir un programa que indique a la máquina los diferentes pasos que debe seguir para obtener los resultados deseados; dicho programa debe escribirse en un lenguaje como el FORTRAN, COBOL, PL/I que es relativamente difícil de aprender, y por esto está reservado a los especialistas.

El BASIC es un lenguaje de programación relativamente nuevo que ha tenido un gran éxito entre los usuarios no especialistas en informática; se desarrolló en el Dartmouth College, por los profesores Kemeny y

Kurtz, adoptado luego por General Electric y también por la mayor parte de los servicios de tiempo compartido. ¿A qué es debido este éxito rápido? Es posible escribir programas sencillos en BASIC con sólo 7 instrucciones (es decir 7 "palabras" del lenguaje). El BASIC es pues muy fácil de aprender: es necesario emplear de cinco a diez horas para aprender el lenguaje y hacer los ejercicios convenientes para adquirir práctica. Por este mismo motivo, es también muy fácil de usar y es el lenguaje ideal para realizar programas de tamaño y complejidad medios. El BASIC puede usarse con o sin tiempo compartido, aunque es especialmente adecuado para la conversación entre el usuario y la máquina, es decir principalmente en tiempo compartido.

El BASIC se dirige pues especialmente:

1. *A los profesionales.* El estudio de un lenguaje de programación representa una inversión en tiempo que no es válida más que con la condición de que este lenguaje se use suficientemente. El estudio del BASIC es tan fácil que se justifica sólo con la escritura de algunos programas de tamaño medio.

2. *A los estudiantes.* Cada vez se les pide más a los estudiantes que conozcan y usen un lenguaje de programación. Como el BASIC es tan potente como la mayor parte de los demás lenguajes y es mucho más simple, se le va escogiendo en un número creciente de Escuelas y Universidades

3. *A los futuros especialistas.* El primer lenguaje de programación es el más difícil de aprender. El BASIC permite familiarizarse con el ordenador antes de emprender el estudio de lenguajes más complejos.

Este libro está destinado a los que desean escribir sus propios programas, así como a todos los que desean conocer cómo es posible dar instrucciones a un ordenador.

Este texto consta de cuatro partes:

I. *Una introducción a la programación*

Esta parte presenta el principio de funcionamiento de un tipo sencillo de ordenador así como un enfoque general a la resolución de problemas con ordenador. El lector que ya tenga algunos conocimientos de informática puede ignorar esta parte y comenzar directamente la lectura del capítulo II.

II. *La definición del lenguaje BASIC*

Se describen las instrucciones del lenguaje y su uso. La lectura de los capítulos "Introducción al BASIC" y "BASIC elemental" basta para escribir programas sencillos. Los capítulos siguientes permiten escribir programas más complejos.

III. *Cómo realizar un programa en BASIC*

Aquí se explica cómo, cuando el usuario ha escrito un programa, puede darlo al ordenador, corregirlo y ejecutarlo.

IV. *Programas útiles*

Esta parte presenta problemas diversos y los programas BASIC que permiten resolverlos con ordenador.

Como final se presenta un primer apéndice con una extensión del BASIC. Las nuevas instrucciones que contiene no forman parte del BASIC original desarrollado en el Dartmouth College, pero están disponibles en la mayor parte de los sistemas actuales. Estas nuevas instrucciones hacen al lenguaje todavía más potente, aunque perdiendo su sencillez. Por esto se aconseja a los que quieran ir más lejos con el BASIC que lean este apéndice solamente después de haber adquirido práctica con el BASIC original.

CAPITULO I

Introducción a la programación

I. EL ORDENADOR

El fin de este capítulo es exponer los conceptos básicos que permiten al usuario de un ordenador, conocer mejor las posibilidades y los límites de su herramienta de trabajo. Se presenta un modelo sencillo de ordenador y un ejemplo ilustrado de su funcionamiento. A partir de este modelo simple, se presentan las posibilidades de sistemas más complejos.

1. ESTRUCTURA DE UN ORDENADOR SIMPLE

Un sistema de ordenador se compone de dos partes fundamentalmente diferentes:

a) El equipo material, llamado *hardware*

b) Un conjunto de programas que permiten el uso de este equipo y que se llama *software.*

El hardware está formado de los cinco elementos siguientes:

1. El *órgano de entrada,* por el que el usuario da las informaciones al ordenador. Los más empleados son las lectoras de fichas, las lectoras de cinta perforada y los teletipos.
2. La *memoria,* en la que el ordenador almacena las informaciones.
3. El *órgano de salida,* por el que el ordenador da los resultados. La impresora y el teletipo son órganos de salida.

4. Un *órgano de cálculo,* que permite hacer operaciones sobre los datos almacenados en la memoria.
5. Un *órgano de control,* que controla a los otros órganos del ordenador.

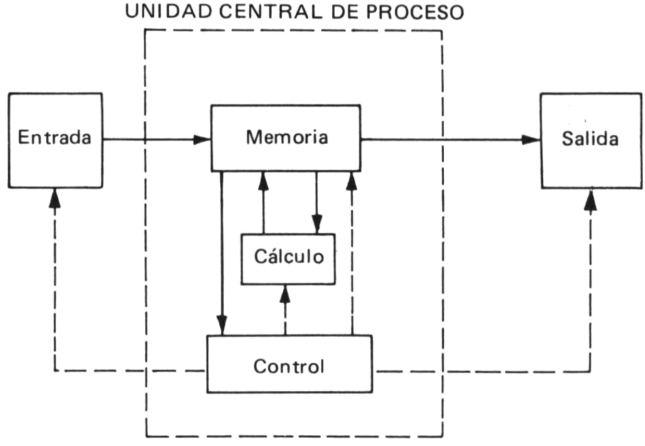


Fig. 1.– (Las líneas de trazo lleno indican *transferencias de datos.* Las líneas de trazo discontinuo representan *señales de control.*)

Al conjunto de Memoria – Organo de cálculo – Organo de control, se le llama *Unidad Central de Proceso.*

Con el siguiente ejemplo se comprenderá mejor el funcionamiento de las diferentes partes mencionadas.

2. FUNCIONAMIENTO DE UN ORDENADOR SENCILLO

Primeramente, vamos a tratar de clasificar el ordenador en la jerarquía de las máquinas. Consideremos un aparato sencillo: un tocadiscos. Para hacerlo funcionar, el usuario lo debe poner en marcha, colocar un disco y poner la aguja sobre el disco; cuando el disco se ha terminado, se debe colocar el brazo del pickup en la posición "Reposo" y retirar el disco. Este ciclo se debe repetir para cada disco que se quiera escuchar. Si se dispone de un tocadiscos *automático* bastará colocar una pila de discos en el eje central y apretar el botón "Marcha". Esto es posible solamente, porque el aparato automático posee un *programa* al que obedece continuamente. Un programa es una serie de instrucciones elementales que la máquina debe ejecutar; el del tocadiscos mencionado puede ser el siguiente:

1. Esperar a que se apriete el botón "Marcha"
2. Dejar caer un disco.
3. Desplazar el brazo para colocar la aguja al comienzo del disco.
4. Esperar a que se termine el disco.
5. Poner el brazo en la posición "Reposo"
6. Si queda algún disco en el eje central, ir al paso 2. En otro caso, ir al paso 7.
7. STOP.

Este programa es fijo y se realiza mediante un sistema de relés electromecánicos contenidos en el tocadiscos.

El ordenador es también una máquina automática. Después de que el usuario apriete el botón "MARCHA", puede funcionar sin ayuda humana y se detiene después de haber dado los resultados. Pero es más que esto: las acciones que toma no están fijadas por un programa interno y rígido, sino por un programa realizado por el usuario para sus propias necesidades. Un ordenador es una *máquina automática programable.*

No basta pues apretar el botón "MARCHA" para hacer funcionar un ordenador. Es necesario también darle un programa indicándole qué acciones deberá realizar. Este programa debe escribirse en un lenguaje especial, siguiendo unas reglas precisas como el ejemplo siguiente, que no es comprensible al lector no iniciado:

```
100     LEER A,B
        HACER  D = A + B
        ESCRIBIR  D
        IR A 100
        FIN
2
4
3
5
```

Cada línea se llama una *instrucción* y corresponde a una orden dada por el usuario al ordenador. El proceso de este programa por la máquina se hace en dos fases distintas:

1. La entrada y traducción del programa a un lenguaje inteligible por la máquina (formado por "0" y "1") y llamado pues "lenguaje máquina". Cuando las instrucciones están escritas en un lenguaje evo-

lucionado (BASIC, FORTRAN, COBOL...), esta fase se llama *compilación.*

2. Los cálculos propiamente dichos, siguiendo las órdenes dadas por el programador. Esta fase se llama *ejecución.*

Antes de ver con más detalle estas dos fases, debemos precisar algo sobre los órganos de la unidad central del proceso.

LA MEMORIA: Puede considerarse como una serie de celdas numeradas de 0 a 12 en nuestro modelo (ver Fig. 2). Cada celda puede contener una cantidad de información tal como: una instrucción elemental (en lenguaje máquina), o bien, un dato numérico.

LA UNIDAD DE CONTROL: Tiene los dos registros siguientes:

– El registro C (o "contador") que contiene el número de la siguiente instrucción a ejecutar.

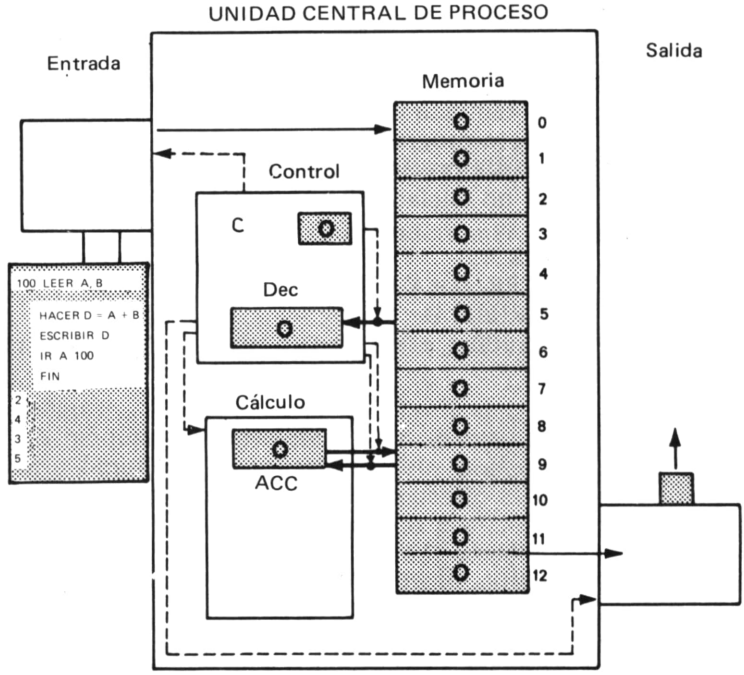


Fig. 2.– (Las líneas de trazo lleno indican *transferencias de datos.* Las líneas de trazo discontinuo representan *señales de control.*)

– El registro DEC ("decodificador") que contiene la instrucción a decodificar. Decodificar significa analizar una instrucción para generar las señales que permiten ejecutarla.

LA UNIDAD DE CALCULO: Posee un registro ACC, el acumulador en el que se hacen las diferentes operaciones entre las variables.

Veamos ahora como el ordenador procesa nuestro programa. La figura 2 representa la figura 1 con más detalle, empleando las precisiones recién mencionadas. Inicialmente la memoria está vacía, es decir, que cada celda tiene ceros. El programa se halla en el órgano de entrada por ejemplo en forma de fichas. El proceso comienza cuando el usuario aprieta el botón "MARCHA" que está en el tablero de mando del ordenador.

a) **Compilación**

Durante la compilación, el ordenador hará principalmente tres cosas:

1. *Traducción de las instrucciones del lenguaje evolucionado al lenguaje máquina.* Este lenguaje máquina se escribe con ayuda de 0 y 1 y cada instrucción es mucho más primitiva que en un lenguaje evolucionado. Una instrucción como la

   HACER D = A + B

   necesitará por ejemplo las siguientes instrucciones de lenguaje máquina:

   ```
   TOMAR      A
   SUMAR      B
   ALMACENAR  D
   ```

   cuyo significado veremos más adelante.

   Estas instrucciones se escriben en binario en la memoria (con ayuda de los únicos caracteres 0 y 1). Aquí las escribimos en "claro" o en forma mnemotécnica, para facilitar la comprensión del proceso.

2. *Detección de las variables y reserva de lugar para las mismas.* Durante la compilación, el ordenador hace una lista de las variables usadas en el programa y a cada una de ellas le asigna una celda de memoria. Cuando la variable A por ejemplo es igual a 5, la celda correspondiente a A contendrá el valor 5.

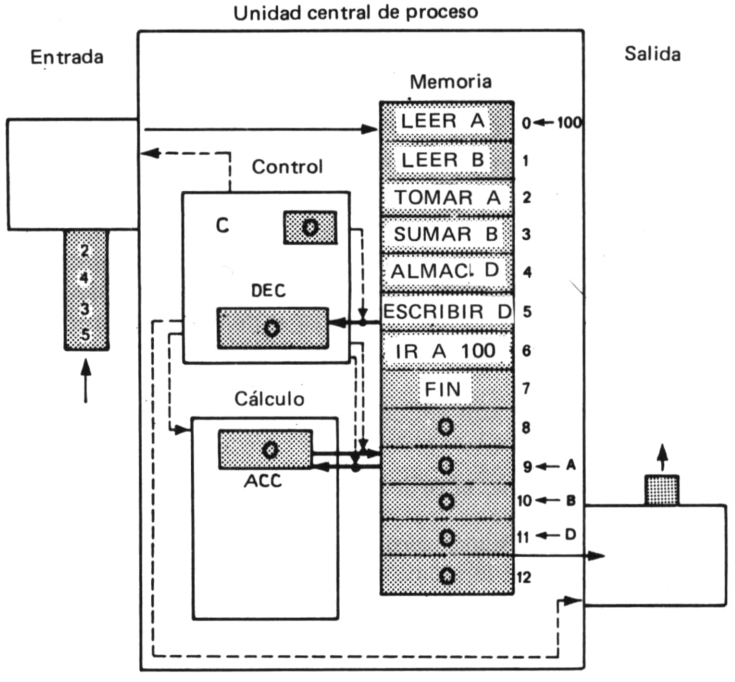


Fig. 3

3. *Detección de las etiquetas.* El número 100 en la instrucción 100 LEER A, B es una "etiqueta". Es necesario que el ordenador sepa a qué celda corresponde esta etiqueta después de la compilación, como se verá a continuación.

Al final de la compilación se han realizado estas tres tareas y el programa queda en la memoria, traducido a lenguaje máquina (ver Fig. 3). El programa está formado ahora por 8 instrucciones en lenguaje máquina (en vez de las 5 en lenguaje evolucionado). Se reservan tres celdas para contener los valores de las variables A, B y D.

La etiqueta 100 corresponde a la celda 0, la lista de datos 2, 4, 3, 5 ha quedado en el órgano de entrada, dispuesta para ser leída.

b) Ejecución

Cuando el ordenador ha leído y traducido la instrucción FIN a lenguaje binario, sabe que ha entrado todo el programa, con lo que la com-

pilación ha terminado por completo. El valor 0 está en el contador C. Comienza la ejecución y el órgano de control va a seguir continuamente el ciclo siguiente:

1. Mirar el contenido del registro C.
2. Poner en el registro DEC la instrucción que se halla en la celda, cuyo número es el contenido de C.
3. Aumentar el contenido de C en 1.
4. Decodificar la instrucción contenida en el registro DEC y generar las señales necesarias para ejecutarla.
5. Recomenzar este ciclo en el paso 1.

Esto va a suceder de la forma siguiente:

1. El ordenador mira en el registro C cuál es el número de la celda que contiene la instrucción a ejecutar. Este número se llama la *dirección* de la instrucción a ejecutar e inicialmente vale 0 (Fig. 4a).
2. El contenido de la celda 0, **LEER A**, se coloca en el registro DEC (Fig. 4b).
3. Se añade 1 a C: C = 0 + 1 = 1 (Fig. 4c).
4. Se decodifica **LEER A**. Se manda una señal por la unidad de control al órgano de entrada para leer el primer dato de la lista y ponerlo en la celda A. Ahora A vale 2 y el primer dato de la lista es 4 (Fig. 4d).

1. El ordenador busca cuál es la dirección de la siguiente instrucción a ejecutar, para lo que mira el contenido del registro C: 1.
2. El contenido de la celda 1, **LEER B** se coloca en el registro DEC.
3. C = 1 + 1 = 2
4. Se decodifica **LEER B**. Como se ha visto antes, el valor 4 se pone en la celda B.

1. C contiene el valor 2.
2. **TOMAR A** se coloca en el registro DEC.
3. C = 2 + 1 = 3
4. Se decodifica **TOMAR A**. Se envían señales para ejecutar esta instrucción, que quiere decir "poner el valor de A en el acumulador

ACC del órgano de cálculo". ACC tiene ahora el valor 2. El contenido de la celda A queda inalterado.

1. C contiene el valor 3.
2. **SUMAR B** se coloca en el registro DEC.
3. C = 3 + 1 = 4.
4. Se decodifica **SUMAR B**: "Sumar el valor de B al valor contenido en el ACC y dejar el resultado en ACC". ACC tiene ahora el valor 2 + 4 = 6. El contenido de la celda B queda inalterado.

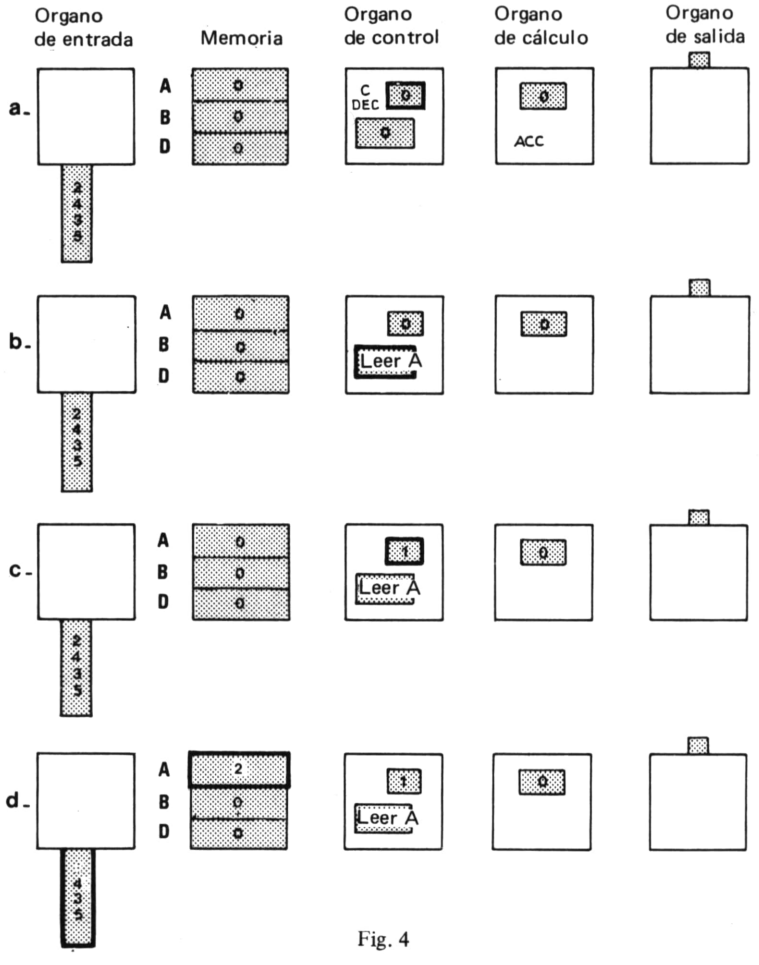


Fig. 4

1. C contiene el valor 4.
2. **ALMACENAR D** se coloca en el registro DEC.
3. C = 4 + 1 = 5.
4. Se decodifica **ALMACENAR D**: "Poner en la celda D un valor igual al contenido en ACC". D ahora tiene el valor 6.

1. C contiene el valor 5.
2. **ESCRIBIR D** se coloca en el registro DEC.
3. C = 5 + 1 = 6.
4. Se decodifica **ESCRIBIR D.** La unidad de control manda una señal al órgano de salida para escribir el valor de D en el teletipo. El valor 6 se imprime. El contenido de la celda D queda inalterado.

1. C contiene el valor 6.
2. **IR A 100** se coloca en el registro DEC.
3. C = 6 + 1 = 7.
4. Se decodifica **IR A 100.** Esta instrucción quiere decir: "la próxima instrucción a ejecutar es la que lleva la etiqueta 100". La ejecución de esta instrucción hace colocar la dirección correspondiente a la etiqueta 100 en el registro C. C valdrá ahora 0.

La próxima instrucción ejecutada será pues **LEER A.** A toma el valor 3. B toma el valor 5. Se calcula D y el valor 3 + 5 = 8 se imprime en el teletipo.

La instrucción **LEER A** se debe ejecutar de nuevo, pero la lista de datos está vacía ahora. ¿Qué pasará? El ordenador no teniendo nada más que calcular, se para sencillamente.

La hoja de resultados contiene los dos números pedidos: 6, 8. Los diversos registros contienen los últimos valores que tomaron (ver Fig. 5).

Este ejemplo nos ha mostrado que no hay nada mágico en un ordenador, sino solamente un conjunto de celdas y de registros entre los que circulan informaciones. Un ordenador es una máquina y no toma ninguna iniciativa: obedece escrupulosamente las órdenes que le da el usuario en forma de programa. Todo error en la especificación de estas órdenes provocará una imposibilidad de funcionar o bien unos resultados erróneos.

Aunque muy simplificado, el modelo de ordenador presentado en estas páginas facilita la comprensión y asimilación del lenguaje de progra-

mación. También es adecuado para referirse a él cuando se esté corrigiendo un programa para comprender mejor la respuesta del ordenador a las instrucciones que se le han dado.

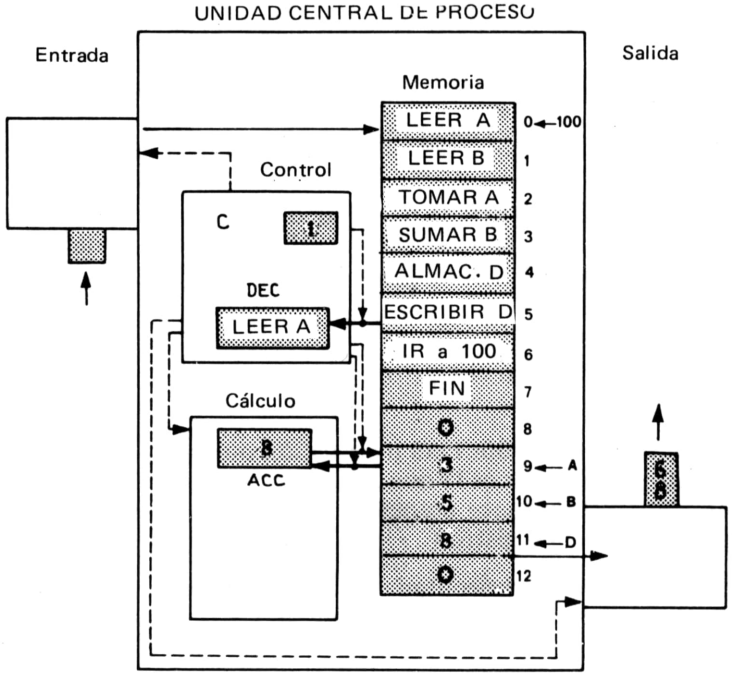


Fig. 5

3. *TIEMPO COMPARTIDO ("TIME SHARING")*

Consideremos un centro de cálculo sencillo que comprenda:

- una unidad central de proceso,
- una lectora de fichas como órgano de entrada,
- una impresora, como órgano de salida,
- una perforadora que permita escribir programas y datos en la forma de fichas para su entrada al ordenador.

Cuando una persona quiere utilizar este centro de cálculo, le basta escribir su programa en forma de fichas, gracias a la perforadora, poner su

paquete de fichas en la lectora y apretar el botón "MARCHA". Los resultados o los mensajes de error salen en la impresora.

Una de las limitaciones esenciales de este sistema es la infrautilización de la Unidad Central de Proceso. En efecto, el ordenador comprende dos partes de naturaleza muy distinta:

1. La Unidad Central de Proceso. Es la unidad más cara del conjunto. Está formada por circuitos electrónicos muy rápidos.
2. Los órganos de Entrada y de Salida que poseen elementos mecánicos, por lo que su velocidad es muy limitada.

El tiempo necesario para leer un dato desde la lectora de fichas es del orden de 0,05 segundos. La Unidad Central de Proceso puede por el contrario ejecutar ciertas instrucciones (por ejemplo la adición de dos números) en menos de 0,000001 segundos. Es decir, que el ordenador podría realizar 50.000 sumas mientras lee un dato. Veamos de nuevo una iteración del programa anterior:

```
LEER      A
HACER     D = A + B
ESCRIBIR  D
```

La ejecución de estas tres instrucciones durará aproximadamente:

- 0,05 segundos para leer A y B (utilizando el órgano de entrada),
- 0,000003 segundos para la decodificación de las tres instrucciones y ejecución de la suma (utilizando la unidad central de proceso).

Estas cifras no son más que estimaciones groseras que dependen en particular de las características tecnológicas de la máquina y de su organización interna. Sin embargo, dan un orden de mangnitud de la utilización de la unidad central durante la ejecución de este programa: ¡0,05 por ciento!

La multiprogramación se ha desarrollado para remediar esta infrautilización de la unidad central. Con esta nueva técnica, se puede partir la memoria entre varios programas. El ordenador ejecuta uno de ellos hasta que encuentra una orden de lectura o escritura. Entonces manda las señales de control al órgano de entrada o de salida y mientras que los datos se leen o se escriben los resultados por un programa, se ejecuta otro programa. De esta manera la unidad central se usa casi continuamente, y el número de programas ejecutados en un tiempo dado es mucho mayor.

La multiprogramación es posible gracias a los desarrollos de hardware y software, demasiado complejos para verlos aquí. Sin embargo, se puede mencionar que un programa especial llamado *sistema operativo* es imprescindible para la gestión de la memoria y de los órganos de entrada y de salida.

El tiempo compartido es una extensión lógica de la multiprogramación: varias personas pueden tener acceso simultáneamente al mismo ordenador. Para ello les basta poseer un órgano de entrada o de salida conectado al ordenador con una línea telefónica. El órgano de entrada/salida se llama en este caso un terminal y el más común es el teletipo, que tiene el aspecto de una máquina de escribir eléctrica y permite entrar con un teclado información en el ordenador y también recibir mensajes del mismo sobre una hoja de papel.

Cada usuario tiene la impresión de que el ordenador trabaja sólo para él. Se puede establecer un diálogo entre el ordenador y el usuario sin que la duración elevada de las lecturas y escrituras suponga un mal uso del tiempo de la unidad central, ya que ésta trabaja para varios usuarios. Un programa que permite este diálogo se le llama "conversacional".

El BASIC se desarrolló para los sistemas de tiempo compartido y en ellos es donde alcanzó su mayor éxito. Sin embargo, también se usa en las instalaciones más tradicionales que no tienen sistemas de tiempo compartido.

EJERCICIOS

1. ¿Qué quieren decir los siguientes términos?
   - hardware
   - software
   - programa
   - compilación
   - ejecución
   - etiqueta
   - multiprogramación
   - tiempo compartido

2. ¿Cuáles son las cinco partes de un ordenador sencillo?

3. ¿Cuáles son los registros principales de un ordenador? ¿Cuál es su papel?

4. En el programa de la página 13 ¿cuál es la particularidad esencial de la instrucción IR A 100 con relación a las otras instrucciones?

5. ¿Qué sucedería si se hubiera omitido la instrucción FIN al compilar el programa de la página 13?

```
100  LEER     A, B
     HACER  D = A + B
     ESCRIBIR D
     IR A  100
2
4
3
5
```

6. ¿Por qué al final de la ejecución del programa de la página 13, el registro C contiene el valor 1 y no el valor 0?

7. Reemplazar la lista de datos 2, 4, 3, 5 del programa de la página 13 por la lista 3, 5, 2. Representar siguiendo el modelo de la figura 4 el estado de los diferentes órganos al *final de la ejecución.*

SOLUCIONES

1. Hardware, software: ver página 11
   Programa: ver página 12
   Compilación, ejecución: ver página 14
   Etiqueta: ver página 16
   Multiprogramación: ver página 21
   Tiempo compartido: ver página 22

2. Ver página 16

3. Ver páginas 14 y 15

4. Aunque las instrucciones de un programa se van siguiendo secuencialmente de arriba hacia abajo, la instrucción IR A permite romper esta secuencia. Esto implica que en la ejecución, la instrucción IR A es la única que modifica el registro C, aparte de los incrementos de uno ya vistos. En el programa de la página 13, esta instrucción sustituye el contenido del registro C por la dirección correspondiente a la etiqueta 100, o sea 0.

5. En este caso, después de compilar IR A 100, el ordenador quiere leer la siguiente instrucción y encuentra el carácter 2 que no es una instrucción sino un dato. El ordenador escribe un mensaje de error, como el "INSTRUCCION INCORRECTA" y no se puede comenzar la ejecución.

6. Veamos de nuevo el último ciclo descrito por la unidad de control:

1. Mirar el valor contenido en C: 0 (el cero lo ha puesto la ejecución de la instrucción **IR A 100**).

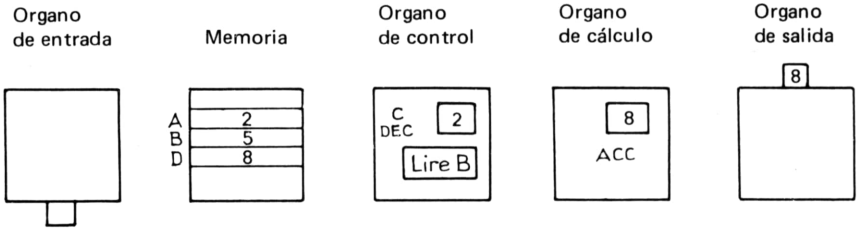
2. El contenido de la celda 0, **LEER A** se pone en el registro DEC.

3. C se aumenta en 1: C = 0 + 1 = 1

4. Se decodifica **LEER A**. Se manda una señal por la unidad de control al órgano de entrada para leer el primer dato de la lista y ponerlo en la celda A, pero la lista está vacía y el ordenador se para.

El registro C contiene el valor 1 aunque la última instrucción decodificada sea la instrucción que se encuentra en la dirección 0. La razón es que *su contenido se aumenta en 1 antes de que se ejecute la instrucción que se halla en el registro DEC.*

7. El ordenador ha calculado la suma de los dos primeros datos 3 + 5 = 8. Ha leído el valor de A: 2 para la suma siguiente. Se ha decodificado la instrucción LEER B, el contenido de C ha aumentado en 1, pero la instrucción LEER B no se ha podido ejecutar por falta de datos.



II. RESOLUCION DE PROBLEMAS CON ORDENADOR

La escritura de un programa, aun en un lenguaje evolucionado como el BASIC o el FORTRAN, puede ser a veces muy difícil. Un cierto número de pasos se deben seguir si se quiere escribir, utilizar y modificar un programa en las mejores condiciones. Aquí se expondrán dichos pasos, así como la herramienta poderosa que son los organigramas en el análisis y la documentación. Por fin, unos conceptos generales sobre la estructura de un programa darán una visión global de las posibilidades que deben suministrar los lenguajes de programación.

1. PASOS A SEGUIR

a) **Definición del problema:**

El programador debe tener una idea exacta del problema a resolver, lo que no siempre es fácil, ya que muy a menudo él trabaja para otras personas y no tiene un conocimiento suficiente de la situación para comprender inmediatamente lo que se espera de él. El problema debe definirse por un diálogo entre los futuros usuarios y el programador. Con frecuencia, un analista se encarga del estudio del problema y da un trabajo más detallado al programador. Esto puede llegar a necesitar varios meses. Por el contrario, puede suceder que una persona tenga un problema sencillo para resolver ella misma, y que ella misma sea capaz de escribir directamente un programa.

De una forma general, se debe definir claramente:

- Cuáles son los resultados buscados y bajo qué forma se quiere presentarlos.
- A partir de qué datos se pueden obtener los resultados anteriores.

b) **Descripción matemática**

Todo resultado se debe obtener a partir de los datos gracias a unas transformaciones matemáticas, que a veces pueden expresarse en forma de fórmulas muy sencillas o por el contrario, necesitar técnicas complejas de investigación operativa. La escritura del programa no puede comenzarse antes de que se hayan definido estas transformaciones.

c) **Programación**

El problema se resuelve en primer lugar en forma de organigramas, que son diagramas, indicando los diferentes pasos que debe seguir el ordenador para llegar al resultado. Su construcción se estudiará en las páginas siguientes cuando el usuario haya podido trazar el organigrama relativo a un problema, éste se considera como teóricamente resuelto. Basta entonces traducir este organigrama a un lenguaje comprensible para el ordenador: BASIC o FORTRAN, por ejemplo.

d) **Ejecución del programa por el ordenador**

e) **Corrección de los errores:**

Es muy raro que un programa sea correcto a la primera escritura: un cierto número de errores se pueden deslizar fácilmente en un programa. Pueden ser de dos tipos:

– *Los errores de sintaxis:* El programador ha escrito una instrucción que no es válida en el lenguaje usado, lo que puede resultar de una distracción, de un fallo de tecleo o simplemente de un mal conocimiento del lenguaje. Este error es descubierto por el ordenador cuando traduce el programa a lenguaje máquina, es decir, durante la compilación. El compilador no comprende lo que el usuario le ordena hacer y ningún cálculo puede comenzarse.

– *Los errores de lógica:* El ordenador ejecuta bien el programa, pero los resultados obtenidos son falsos. Las órdenes dadas por el programador al ordenador no le permiten resolver el problema dado. Es un error de concepto.

La corrección de errores ocupa un tiempo a menudo comparable al tiempo de escritura del programa.

f) **Documentación:**

Cuando el programa funciona correctamente, se debe hacer una documentación para:

– Permitir a otras personas su uso.
– Poder modificarlo en el futuro si fuera necesario.

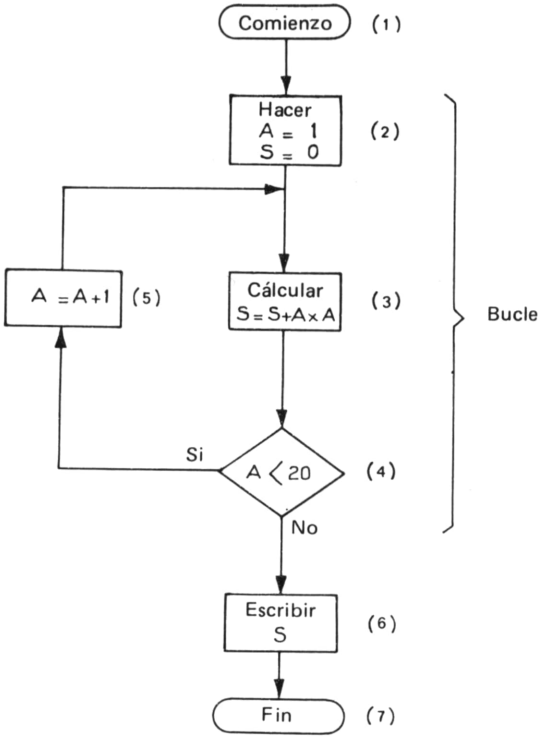
2. LOS ORGANIGRAMAS

Los organigramas son una herramienta poderosa en programación. Permiten descomponer la resolución de un problema en elementos para facilitar la escritura de un programa; constituyen igualmente una parte esencial de la descripción de un programa en una documentación.

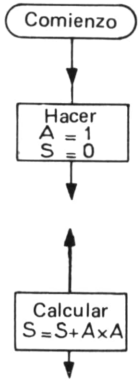
Con un ejemplo sencillo introduciremos los principales signos usados para construir organigramas. Supongamos que se tenga que calcular la suma de los cuadrados de los veinte primeros números enteros:

$$S = 1 \times 1 + 2 \times 2 + 3 \times 3 + 4 \times 4 + \ldots + 20 \times 20$$

Esta suma se puede obtener calculando 1 x 1, luego se suma a este resultado 2 x 2 = 4, lo que da 5; a este resultado parcial se le suma 3 x 3 = 9, resultando 14; y así hasta llegar a calcular 20 x 20 y añadiéndolo a la suma de los cuadrados de los 19 primeros números enteros.



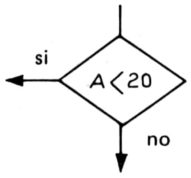
Este procedimiento se puede representar con el organigrama superior:



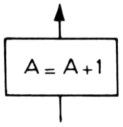
Indica el comienzo del procedimiento.

Es la inicialización del proceso. La variable A contiene el número que se debe elevar al cuadrado, o sea 1.
La variable S contiene la suma acumulada de los cuadrados y vale 0 al comienzo.

A se eleva al cuadrado y se adiciona a la antigua suma acumulada S para tener el nuevo valor de S.



Esto es una prueba. Sólo se deben elevar al cuadrado y sumar los 20 primeros números enteros. Si A es inferior a 20, S es una suma parcial y los cálculos deben seguir. Si por el contrario A es igual o superior a 20, S es la suma final y se han terminado los cálculos.

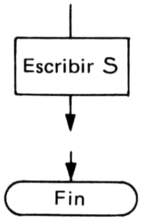


A se incrementa en 1 para dar el valor del siguiente número entero a elevar al cuadrado.

Los tres símbolos últimos forman parte de un mismo bucle en el organigrama. Este bucle se recorrerá veinte veces, para los valores 1, 2, 3, 4, ..., 20 de la variable A.

Cada vez el valor de A se elevará al cuadrado y se sumará al total acumulado de S, dando

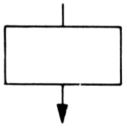
$$S = 1 \times 1 + 2 \times 2 + 3 \times 3 + \ldots + 20 \times 20$$



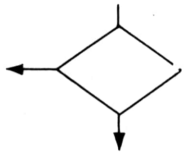
El valor de S se imprime en la hoja de resultados.

Indica el fin del procedimiento.

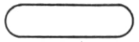
Este ejemplo sencillo nos ha permitido introducir los tres símbolos más usados en los organigramas:



Indica un cálculo, una escritura o una lectura.



Indica una prueba que permite escoger entre 2 alternativas posibles.

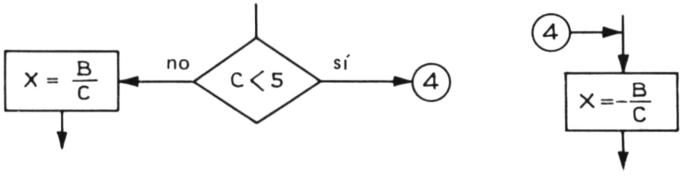


Indica un extremo del procedimiento: comienzo, final o interrupción.

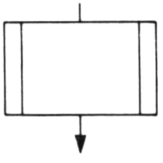
Estos símbolos se enlazan con flechas que indican la secuencia de las operaciones. Los detalles relativos a los mismos se escriben en el interior de los símbolos.

Entre los otros símbolos empleados en los organigramas es conveniente mencionar:

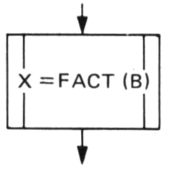
Símbolo de unión entre dos puntos de un organigrama, por ejemplo:



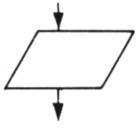
Si C es menor que 5, entonces X toma el valor –B/C. Con este símbolo se obtiene una mejor presentación de los organigramas.



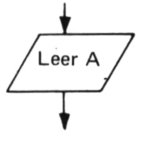
Indica un proceso o tratamiento definido en otra parte, por ejemplo:



en donde el tratamiento FACT se ha definido anteriormente.



Símbolo usado a veces para indicar una orden de lectura o escritura.



A menudo es preferible usar el mismo símbolo para las órdenes de lectura y escritura que para el proceso (es decir, el rectángulo). Así se hizo en el organigrama visto antes.

3. ESTRUCTURA GENERAL DE UN PROGRAMA

De la misma manera que las lenguas vivas como el castellano, francés, inglés tienen semejanzas en su estructura (nombre, verbo, adjetivo, artículo, ...), así también los lenguajes evolucionados de programación poseen todos instrucciones que permiten escribir las diferentes fases de un programa.

a) **Declaraciones**

Con ellas se indica al ordenador que deberá reservar memoria para ciertas variables empleadas en el programa.

b) **Lectura**

En la mayoría de los casos, la resolución de un problema requiere la introducción en memoria de datos, lo que se realiza mediante órdenes de lectura.

c) **Proceso**

Es la parte que permite resolver el problema. Las órdenes de proceso son, por ejemplo: cálculos de expresiones, asignación de valores a las variables, las transferencias de control...

d) **Escritura**

Los resultados obtenidos en la fase de proceso se imprimen con órdenes de escritura.

e) **Datos**

Los datos se pueden meter en el programa, pero otras alternativas (formación de ficheros, entrada por el usuario durante la ejecución) son preferibles con frecuencia.

Estas diferentes fases no son en ningún caso independientes, ni siquiera físicamente distintas en un programa: por ejemplo las órdenes de lectura o de escritura pueden encontrarse en la parte de proceso. Estas fases pueden asimismo tener importancias relativas muy distintas según

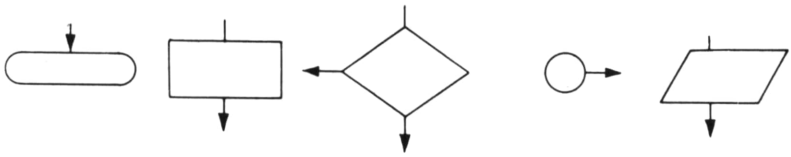
el tipo de problema: así los problemas científicos necesitan en general muchos procesos y pocas lecturas. Por el contrario, los problemas de gestión precisan pocos procesos, pero muchas lecturas y escrituras.

El lenguaje BASIC que se presenta en las páginas siguientes da la posibilidad de construir muy fácilmente un programa que contenga estas diferentes fases con un mínimo de instrucciones.

EJERCICIOS

1. Indicar las seis fases necesarias para la realización de un programa.

2. ¿Qué representan los símbolos siguientes en un organigrama?



3. Qué haría falta cambiar en el organigrama de la página 27 si se quisiera calcular:
   a) La suma de los cuadrados de los números enteros de 4 a 20: $4 \times 4 + 5 \times 5 + ... + 20 \times 20$;
   b) La suma de los cuadrados de los números impares de 1 a 20: $1 \times 1 + 3 \times 3 + ... + 19 \times 19$;
   c) La suma de los cuadrados de los números pares de 1 a 20: $2 \times 2 + 4 \times 4 + + 6 \times 6 + ... + 20 \times 20$;
   d) El producto de los números enteros de 1 a 20: $1 \times 2 \times 3 \times 4 \times ... \times 19 \times 20$.

4. Trazar un organigrama que represente el cálculo de la media y de la varianza de una serie de N datos estadísticos. Estos datos deben ser leídos por el ordenador en la ejecución del programa:

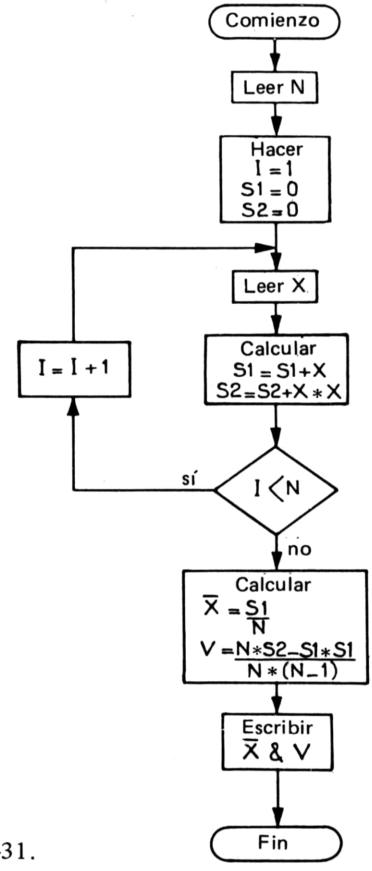
$$\text{media } \bar{X} = \frac{\Sigma X_i}{N}$$

$$\text{varianza } V = \frac{N\Sigma X_i^2 - (\Sigma X_i)^2}{N(N-1)}$$

5. ¿Cuáles son las cinco clases de información que un programa debe contener?

SOLUCIONES

1. Ver las páginas 24 - 26
2. Ver las páginas 27 - 29
3. a) Cambiar en (2) A = 1 por A = 4
   b) Cambiar en (5) A = A + 1 por A = A + 2
   c) Cambiar en (2) A = 1 por A = 2
      en (5) A = A + 1 por A = A + 2
   d) Cambiar en (3) S = S + A x A por S + S x A
      Cambiar en (2) S = 0 por S = 1
4. El organigrama siguiente representa una de las formas de calcular la media y la varianza de una serie de N datos estadísticos.



5. Ver las páginas 30-31.

CAPITULO II

Definición del lenguaje BASIC

El lenguaje de programación BASIC ha sido desarrollado por los profesores John G. Kemeny y Thomas E. Kurtz en el Dartmouth College. Igual que el Fortran, el Algol y el Cobol, también el Basic es un lenguaje de alto nivel y se diferencia, sin embargo, de estos lenguajes por las dos importantes propiedades:

1. Una mayor sencillez y facilidad de uso. Prácticamente con sólo siete instrucciones se puede escribir cualquier programa en Basic.
2. Una mejor adaptación al diálogo hombre-máquina en el tiempo compartido.

El Basic se desarrolló primero para el sistema GE 265, empleando un ordenador GE 235 y luego para la mayor parte de los ordenadores de la empresa General Eléctrica. Esta empresa ha introducido el Basic en su servicio de tiempo compartido y el lenguaje ha sido adoptado por un gran número de usuarios que rápidamente reconocieron sus méritos. Actualmente, la mayor parte de los servicios de tiempo compartido permiten el uso del Basic. Se han desarrollado varios intérpretes (una especie de compiladores) para traducir programas de BASIC a FORTRAN, permitiendo de esta forma la utilización del BASIC en todo ordenador de tamaño medio o grande que disponga de un compilador FORTRAN.

Gracias al desarrollo actual de los servicios de tiempo compartido y a su facilidad de uso, el lenguaje BASIC ha experimentado un éxito muy rápido, que parece increible.

I. INTRODUCCION AL BASIC

Comencemos con un ejemplo muy sencillo:

```
100  REM  FACTURACIØN
110  READ N
120  READ P
130  READ H
140  READ M
150  READ T
160  LET C = N * P
170  LET C = C + M * H
180  LET C = C * (1 + T)
190  PRINT C
200  GØ TØ 110
210  DATA 2,4.52,1.5,9.5,0.23456
220  DATA 3,11,5.75,12,0.17647
230  END
```

Cada línea del programa contiene un número (el "número de línea") y una instrucción. Por ejemplo: 160 LET C = N * P contiene el número de línea (160) y una instrucción (LET C = N * P).

La disposición de los caracteres sobre una línea no tiene ninguna importancia, sólo su orden debe respetarse. Así:

```
160  LET C = N * P
160L   E   TC = N * P
16     0    LE   TC   = N * P
```

son equivalentes.

Los números de las líneas deben ser números enteros comprendidos entre 0 y 99999, y deben ir en orden ascendente de arriba a abajo. Se prefiere en general dar a las líneas números no consecutivos: aquí se ha adoptado un intervalo regular de 10 entre cada número, lo que permite insertar una línea en el programa sin tener que cambiar los números siquientes. El número de la primera línea es arbitrario y por supuesto puede ser diferente de 100.

Este programa permite calcular el importe de una factura hecha por un garaje para sus reparaciones. Los elementos necesarios para el cálculo son:

- El precio unitario de la pieza cambiada. Se supone para simplificar que sólo hay un tipo de pieza.
- El número de piezas cambiadas.

- El número de horas de trabajo.
- El coste horario de la mano de obra.
- La incidencia del IVA (Impuesto Valor Añadido) sobre el coste total de las piezas y de la mano de obra.

La primera instrucción es un comentario. Se escribe REM y después puede llevar cualquier comentario. El ordenador lo ignora y sirve sólo para recordar algo al programador o dar una explicación al usuario del programa.

La instrucción siguiente permite introducir la noción de variable. En la primera parte del libro hemos visto que una variable podrá considerarse como es el nombre de una celda en la que el ordenador pone valores que usará enseguida en los cálculos. En cada lenguaje de programación existen convenciones para definir una variable. En Basic, una variable puede definirse con una letra o bien una letra seguida de un dígito (0 a 9):

A, C, Z, I, T, X3, Ø2, R9 son variables (escribimos Ø para la letra O para diferenciarla del número 0).

3A, X13, TØ, ZA, 4, B11 no son variables, y su empleo impedirá que un programa funcione.

Por supuesto, las variables que representan magnitudes diferentes deben escribirse distintamente.

En las líneas 110 a 150, las instrucciones de lectura READ N, READ P, READ H, READ M y READ T, permiten dar valores a las variables N, P, H, M y T.

Estas variables representan:

N = número de unidades cambiadas,
P = precio unitario de la pieza cambiada,
H = número de horas de trabajo,
M = coste horario de la mano de obra,
T = incidencia del IVA (Impuesto Valor Añadido).

A las instrucciones READ deben siempre corresponder en un programa por lo menos una instrucción DATA (líneas 210 y 220). DATA debe ir seguido de datos separados entre sí por comas. Los datos pueden ser enteros (p. ej. los 2;3;11;12) o reales (p. ej. los 1,5;9,5;0,23456). En este último caso se deben escribir con un punto en vez de la coma decimal, siguiendo la costumbre anglosajona:

1.5, 9.5, 0.23456 representa una serie de números reales: 1,5; 9,5 y 0,23456, mientras que 1, 5, 9, 5, 0, 23456 representa una serie de seis números enteros: 1; 5; 9; 5; 0 y 23456.

El conjunto de los números que aparecen después de las instrucciones DATA se llama una lista de datos: ella comienza en el primer número de la primera instrucción DATA y se termina en el último número de la última instrucción DATA. En nuestro programa la lista de datos es:

2; 4.52; 1.5; 9.5; 0.23456; 3; 11; 5.75; 12; 0.17647

La instrucción general READ X es equivalente a la orden siguiente dada al ordenador: "dar el primer valor *aún no utilizado* de la lista de datos a la variable X". Las instrucciones 110, 120, 130, 140 y 150 darán el valor 2 a N, 4.52 a P, 1.5 a H, 9.5 a M y 0.23456 a T. Es pues necesario cuando se forme la lista de datos, el colocarlos en un orden tal que las variables tomen los valores deseados y no otros.

En este punto, la fase "entrada" ha terminado y ya pueden comenzar los cálculos, que se realizan gracias a la instrucción LET. El signo * en la línea 160 representa una multiplicación, ya que el signo de multiplicación x que se confunde fácilmente con la letra x, no se usa en Basic. La instrucción LET C = N * P es equivalente a la orden siguiente:

"Calcular el producto N * P y dar después a C el valor resultante". Una instrucción LET debe ir siempre seguida de una variable, de un signo igual y de una expresión que puede contener variables, signos operativos y constantes. La misma variable puede encontrarse a la izquierda y derecha del signo igual, como en la instrucción:

```
170   LET C = C + M * H
```

lo que parece ridículo si se le da al signo igual el mismo sentido que en álgebra. Se tendría entonces:

C = C + M * H luego 0 = M * H

De hecho, recordando la definición que se acaba de hacer, esta instrucción quiere decir: "Calcular la expresión C + M * H luego dar a la variable C el valor resultante". El valor de C calculado en 160 se emplea para calcular C + M * H, *luego* C toma un nuevo valor igual al resultado del cálculo. El signo igual usado en una instrucción LET puede traducirse por "toma el valor de ": LET A = B * C quiere decir "A toma el valor de B * C". La instrucción 180 se explica de la misma forma.

¿Qué representan estos cálculos?

La variable C es el precio total que deberá pagar el cliente. En 160, se calcula el valor de las piezas cambiadas (N * P). En 170 se le añade el coste de la mano de obra (M * H), y en 180 se multiplica el total anterior por 1 + T para incluir el IVA. Después de 180, C es el total a pagar incluyendo impuestos. La fase de cálculo ha terminado. El resultado debe ser comunicado por el ordenador al usuario del programa, lo que se realiza gracias a una instrucción de escritura: PRINT C en la línea 190. Ella tiene como efecto el imprimir el valor final de C sobre la hoja de papel del teletipo.

El usuario del programa posee pues ahora el precio total correspondiente a los datos de la línea 210.

El escribir un programa para obtener este resultado no parece muy útil. Este programa no será interesante más que si se repite un gran número de veces, lo que se realiza gracias a la línea 200 GØ TØ 110 que equivale a decirle al ordenador "Ir a la línea 110, ejecutar la orden que contiene y seguir secuencialmente a partir de esta línea". En la línea 110 comienza la fase de entrada y el ciclo de entradas/cálculos/salidas se repetirá varias veces, ya que el ordenador ejecuta instrucciones que forman un bucle por lo que "girará sobre su eje". De hecho este bucle contiene órdenes de lectura y en cada ciclo se van a tomar valores de la lista de datos. Cuando haya usado todos los valores de la lista de datos y necesite un valor adicional, el ordenador se parará e imprimirá el mensaje "ØUT ØF DATA", como se puede ver en la hoja de resultados.

```
28.7528
119.999
ØUT ØF DATA AT 110
```

El ordenador ejecuta un programa mientras se le suministre trabajo, es decir, datos, y no parece útil indicarle cuándo debe pararse. De hecho, es más fácil y elegante el indicárselo, como veremos más adelante.

La última línea de un programa en Basic debe siempre contener la instrucción END para indicarle al ordenador dónde debe pararse cuando esté traduciendo el programa Basic a lenguaje máquina.

RESUMEN.– Gracias a un programa sencillo, nos hemos encontrado las instrucciones necesarias para formar un ciclo elemental de entradas/cálculos/salidas.

Estas instrucciones son:

READ y DATA
LET
GØ TØ
PRINT
END

El programa descrito puede modificarse considerablemente para aumentar su eficacia y los servicios que nos puede dar. Ya veremos cómo en las próximas páginas.

EJERCICIOS

1. ¿Qué variables son ilegales? ¿Por qué?

X3	LX	9	ANS
6K	Y12	D	3L2
O3	Ø5	DD	LØ

2. ¿Qué expresiones son ilegales? ¿Por qué?

Y(D − L)	Z + 2.
E * C − B3	A + 4 × C

3. Encontrar un error en el programa siguiente:

```
100  READ A,B,C
110  LET S = (A + B) * C
120  PRINT S
130  END
```

¿Qué hará el ordenador al ejecutar este programa?

4. Encontrar un error en el programa siguiente:

```
100  READ X,Y
110  LET T = 3 * X + Y
120  LET T = T * C
130  PRINT T
140  DATA 1,2
150  END
```

¿Se obtendrá un resultado? Si la respuesta es sí ¿qué valor de T se imprimirá?

5. Cómo será la hoja de resultados del programa siguiente:

```
100  READ A1,A2
110  LET N = A1 + A2
120  PRINT N
130  GØ TØ 100
140  DATA 1,3,5,3.5,2.8,4,9.3
150  END
```

SOLUCIONES

1. Las variables siguientes son ilegales:

LX
9
ANS
6K
Y12
3L2
O3
DD
LØ

El nombre de una variable debe ser:
- una letra, o
- una letra seguida de una cifra (0 a 9)

2. Las expresiones siguientes son ilegales:

   Y (D - L) el signo * se debe escribir siempre cuando se quieren multiplicar dos factores. La expresión Y * (D - L) es correcta.

   A + 4 x C ya que el signo "x" no existe en Basic. La expresión A + 4 * C es correcta.

3. No hay instrucción DATA que permita ejecutar la instrucción READ de la línea 100. El ordenador imprimirá el mensaje siguiente:

```
ØUT ØF DATA AT 100
```

   y abandonará la ejecución del programa.

4. La variable C que aparece en la línea 120 no ha sido definida. Cuando se llegue a la línea 120, el ordenador no sabrá qué valor de C tomar para efectuar el cálculo. En este caso, dará arbitrariamente el valor 0 a C, con lo que T será igual a 0 y éste es el valor que se imprimirá.

5. La hoja de resultados será:

```
4
8.5
6.8
ØUT ØF DATA AT 100
```

II. BASIC ELEMENTAL

El ejemplo presentado en el capítulo precedente ha permitido introducir los elementos principales del lenguaje Basic. Este capítulo desarrolla y completa estos elementos estudiando sucesivamente las instrucciones de entrada y salida, las que permiten las transferencias de control y la construcción de bucles.

1. ENTRADA

Existen dos instrucciones que permiten la entrada de los datos: READ e INPUT.

a) La instrucción READ

Ya la hemos encontrado en nuestro ejemplo. Recordemos que permite atribuir a las variables unos valores contenidos en una lista de datos precedidos por la instrucción DATA. Estos datos deben estar especificados por el usuario antes que el programa entre en memoria.

En nuestro programa tenemos:

```
110 READ N
120 READ P
130 READ H
140 READ M
150 READ T
```

lo que parece indicar que se necesita una instrucción READ para cada dato que se quiere leer, lo que sería bastante incómodo. Afortunadamente, de la misma manera que se puede formar una lista de datos después de una instrucción DATA, se puede formar una lista de variables después de una instrucción READ.

Las líneas 110 a 150 pueden pues escribirse más sencillamente en una sola línea:

```
110 READ N,P,H,M,T
```

Las instrucciones DATA pueden hallarse en cualquier sitio del programa, antes o después de la instrucción READ correspondiente. En realidad sólo el orden en que estén influye en el resultado.

Es una buena regla el agrupar los datos al final del programa, justo antes de la instrucción END, para tener así más claridad, poder modifi-

car más fácilmente los datos y disminuir los riegos de error. Por las mismas razones, se hace corresponder a menudo una instrucción DATA a una sola instrucción READ. Por ejemplo:

```
120  READ A,B
      .
      .
210  READ L,M,N,Ø,P,Q
      .
      .
350  DATA 6,14.3
360  DATA 2.3,14,172,0.42,0.7,21
370  END
```

La línea de datos 350 corresponde a la orden de lectura 120 y la línea de datos 360 corresponde a la orden de lectura 210.

No hay límite al número de instrucciones DATA presentes en un programa.

Todos los datos empleados en los ejemplos anteriores están escritos en forma entera o decimal. Pero es posible usar otra notación exponencial, por ejemplo, 3:2 E 3 que se lee: 3,2 multiplicado por 10 a la potencia 3, o sea $3{,}2 \times 10^3 = 3{,}2 \times 1000 = 3200$.

De la misma forma:

```
452 E - 2 = 4.52
198.34 E1 = 1983.4
- 2.5986 E - 1 = - 0.25986
```

A continuación se da un ejemplo de uso de la notación exponencial:

```
100  READ A,B,C
110  LET D = 10 * A + B * C
120  PRINT D
130  DATA 3.2E1, 8.5, 45E2
140  END
```

b) **La instrucción INPUT**

Permite al usuario especificar el valor de los datos, pero sólo durante la ejecución del programa. La instrucción INPUT va seguida de una lista de variables, como la instrucción READ.

Por ejemplo:

```
100  INPUT A,B,C
```

Cuando el ordenador se vuelva a encontrar esta instrucción imprimirá un signo de interrogación para indicar al usuario que debe teclear los datos. Los datos deberán ir separados por comas. He aquí un ejemplo de un programa que emplea una instrucción INPUT:

```
100  INPUT A,B,C
110  LET S = A + B + C
120  PRINT S
130  END
```

que dará lugar a la conversación siguiente entre el ordenador y el usuario.

```
?              (ordenador)
3,4 . 5,2      (usuario)
9 . 5          (ordenador)
```

Un mismo programa puede, por supuesto, contener instrucciones READ e instrucciones INPUT.

EJERCICIOS

1. ¿Qué valor se imprimirá?

```
 50  DATA 5
 55  READ A,B,C
 60  DATA 2.5
 70  LET D = B * C - A
 80  PRINT D
 90  DATA 4
100  END
```

2. Se quiere dar a las variables X, Y, Z del programa los valores respectivos 4, 3 y 8.

```
100  READ X,Y
110  INPUT Z
120  LET L = Z - Y * X
130  PRINT L
140  END
```

Encontrar qué instrucción falta. ¿Cuándo se tecleará el valor de Z en el teletipo?

SOLUCIONES

1. El valor impreso por el ordenador será 5. En efecto, A = 5, B = 2.5; C = 4. De donde D = 2.5 x 4 - 5 = 5.

2. La instrucción DATA correspondiente a la instrucción de lectura de la línea 100 falta. Se puede incluir por ejemplo antes de la instrucción END: 135 DATA 4,3.

   El usuario deberá teclear el valor 8 de Z, después de que el ordenador imprima un signo de interrogación.

2. SALIDA

En el programa FACTURACION, la instrucción PRINT se ha empleado para imprimir el valor de una variable.

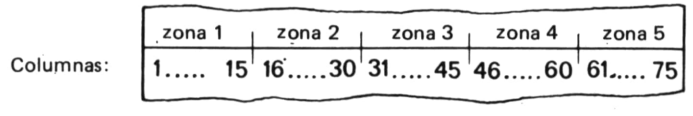
Ella permite igualmente:

- Escribir el valor de una lista de variables.
- Escribir el valor de una expresión.
- Escribir comentarios.
- Realizar la presentación deseada sobre la hoja de resultados.

La instrucción PRINT puede, como la instrucción READ, ir seguida de una lista de datos, como por ejemplo:

```
90PRINT A3,B,M,K2,Z
```

Las variables impresas por la misma instrucción PRINT estarán sobre la misma línea de impresión. Para esto, cada línea de la hoja de resultados está dividida en 5 zonas de un ancho de 15 caracteres cada una, y cada variable se imprime a partir del comienzo de una zona:



El primer carácter de A3 se imprimirá así en la columna 1, el de B en la columna 16, etc. Sólo se pueden imprimir pues 5 números en una

misma línea. ¿Qué sucederá si se tiene una lista de más de 5 variables? Que después de haber impreso la quinta variable, el ordenador pasa automáticamente a la línea siguiente y continúa siguiendo la misma ley.

La coma en 60 PRINT A, B puede considerarse como que fuera la orden siguiente dada al ordenador: "saltar al comienzo de la zona siguiente de 15 caracteres (o al comienzo de la zona de la próxima línea, si ésta estuviera ya llena), y esperar la orden siguiente". La próxima orden es aquí imprimir el valor de B. La ausencia de coma después de B indica al ordenador que puede abandonar la línea sobre la que ha impreso B y saltar a la línea siguiente. Consideremos los dos grupos de instrucciones siguientes:

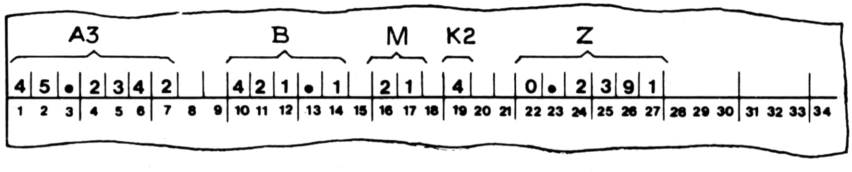
```
\ 50  PRINT A          \ 50  PRINT A,
/ 60  PRINT B    y     / 60  PRINT B
```

En el primer caso A y B se imprimen en dos líneas diferentes, mientras que en el segundo lo hacen en la misma línea, gracias a la coma que va al final de la instrucción 50.

Se dispone de otra alternativa para agrupar más los resultados en una misma línea; en efecto, se puede escribir:

```
90  PRINT A3;B;M;K2;Z
```

En este caso, en lugar de dividirse la línea en zonas de 15 caracteres, cada línea se divide en zonas de 3 caracteres. El primer carácter de B se imprimirá así al comienzo de la primera zona de 3 caracteres *enteramente* libre después de A3. Se tendrá por ejemplo:



Un punto y coma al final de una instrucción PRINT permitirá a la próxima instrucción PRINT el imprimir un valor sobre la misma línea si no está llena.

Los números impresos con una instrucción PRINT tienen, en la mayor parte de sistemas, un máximo de 6 cifras. Así, p. ej. 24.286524 será redondeado y escrito como 24.2865. Cuando los números son o muy grandes o muy pequeños, se escriben en forma exponencial.

El valor de una expresión puede imprimirse directamente en la hoja de resultados mediante una instrucción PRINT:

```
75  PRINT A * (B + C), K
```

que imprimirá el valor de la expresión A * (B + C) y el valor de la variable K en la misma línea. En este caso, el ordenador ha calculado el valor de A * (B + C) sólo para imprimirlo, sin conservarlo en memoria. Si se quiere usar este valor otra vez, habrá que calcularlo de nuevo. Esto explica por qué se prefiere a menudo dar a una variable el valor de la expresión antes de imprimirla, como sigue:

```
70  LET D = A * (B + C)
75  PRINT D,K
```

Hemos visto como una instrucción PRINT permite la escritura de resultados. Para aumentar la facilidad de uso de los mismos, es necesario tener la posibilidad de indicar su significado en claro al usuario; lo que se realiza igualmente con la instrucción PRINT. Por ejemplo, 190 PRINT "NETO A PAGAR TTC:", C dará la impresión:

```
NETO A PAGAR TTC:          28.7528
```

Todo comentario que vaya entre comillas y después de una instrucción PRINT, el ordenador lo imprime *enteramente*. Antes hemos mencionado que la presencia de blancos en una línea no tenía ninguna importancia. La única excepción es en el caso de un comentario a imprimir: si se quiere un espacio en la hoja de resultados entre A y PAGAR, deberá figurar en el programa. Como en general es difícil de ver y de contar los blancos en un manuscrito, es conveniente indicarlos con un ␣ cuando se escribe un programa:

```
70  PRINT "␣ NUMERO ␣ DE ␣ PIEZAS ␣ : ␣", N
```

Cuando se quiere escribir el valor de una variable después de un comentario, se tienen tres posibilidades; se puede imprimir:

– Inmediatamente después del comentario, para lo que basta *no* separar el nombre de la variable del comentario con ningún signo de puntuación en la instrucción PRINT:

```
50  PRINT "A =" A
```

– Al comienzo de la primera zona de tres columnas enteramente libre después del comentario, introduciendo un *punto y coma* entre la comilla y el nombre de la variable:

```
50  PRINT "A = "; A
```

– Al comienzo de la primera zona de 15 columnas enteramente libre después del comentario, introduciendo una *coma* entre la comilla y el nombre de la variable:

```
50  PRINT "A = ", A
```

Las tres posibilidades se muestran en el ejemplo siguiente:

```
10  READ A
20  PRINT "A =" A
30  PRINT "A ="; A
40  PRINT "A =", A
50  DATA 5
60  END
```

que imprimirá sobre la hoja de resultados:

```
A = 5
A =    5
A =                                    5
```

La instrucción PRINT permite igualmente el centrar horizontalmente y verticalmente los resultados o los comentarios:

```
90  PRINT "␣␣␣␣␣␣␣";A ; "␣␣␣␣␣␣␣" ; B
```

Si A y B son enteros inferiores a 999, esta instrucción va a colocar A en las columnas 7, 8 y 9 y a B en las columnas 16, 17 y 18.

La secuencia:

```
90  PRINT A
95  PRINT
100 PRINT B
```

permite imprimir A y B separados por una línea de blancos.

EJERCICIOS

1. ¿Qué hace el programa siguiente?

```
100  PRINT "ESCRIBIR LOS VALORES DE A y C"
110  INPUT A,C
120  PRINT A * C
130  END
```

2. Describir la hoja de resultados del programa siguiente:

```
100  READ A
110  PRINT "A =";
120  READ B
130  PRINT A, "B ="; B, "A * B =" A * B
140  DATA 5,8
150  END
```

3. Se quiere mejorar el programa FACTURACION para que imprima directamente una factura. Hace falta otra variable C, el número del cliente. El programa es:

```
120  REM FACTURAS
130  READ C,N,P,H,M,T
140  LET R1 = N * P
150  LET R2 = M * H
160  LET R3 = R1 + R2
170  PRINT
180  PRINT
190  PRINT "                    CLIENTE NØ";C
200  PRINT
210  PRINT "PIEZAS DE RECAMBIO"
220  PRINT "    "; N; "A"; P; "$                    "; R1; "$"
230  PRINT "MANO DE OBRA"
240  PRINT "    "; H; "HORAS    A"; M; "$          "; R2; "$"
250  PRINT
260  PRINT "TØTAL ANTES IVA                        "; R3; "$"
270  PRINT "IVA (INCIDENCIA"; T;")              "; R3 * T; "$"
280  PRINT
290  PRINT "NETO A PAGAR TTC                    "; R3 * (1 + T); "$"
300  PRINT
310  PRINT
320  GØ TØ 130
330  DATA 0129,2,4.52,1.5,9.55,0.23456
340  END
```

¿Cuál es el formato de la factura que se obtiene con este programa?

SOLUCIONES

1. La hoja de resultados de este programa es:

```
ESCRIBIR LOS VALORES DE A y C     (ordenador)
?                                 (ordenador)
4,5                               (usuario)
20                                (ordenador)
```

Cuando se usa la instrucción INPUT, se le hace preceder en general de una instrucción PRINT indicando al usuario cuáles son los valores que debe teclear.

2. La hoja de resultados de este programa es:

A = 5 B = 8 A * B = 40

3. La factura obtenida tiene la forma siguiente:

```
                    CLIENTE NØ 129

PIEZAS DE RECAMBIO
   2      A 4.52 $                    9.04 $
MANO DE OBRA
   1.5 HORAS A        9.55 $          14.325 $

TØTAL ANTES IVA                       23.3650 $
IVA (INCIDENCIA .23456)                5.48049 $

NETO A PAGAR TTC                      28.8454 $

ØUT ØF DATA AT 130
```

3. CALCULOS

Se realizan con instrucciones LET que ya hemos usado para hacer multiplicaciones y sumas y que permite calcular cualquier clase de expresiones.

Recordemos primeramente que el signo "=" empleado en LET C = = N * P no tiene el mismo significado que en álgebra, sino que quiere decir "C toma el valor N * P".

Esto explica en particular:

– Que se pueda tener una instrucción del tipo LET C = C + M * H con C a los dos lados del signo "=", queriendo decir "C toma como nuevo valor su antiguo valor más el producto M * H".

– Que se deba tener una sola variable a la izquierda del signo "=".

La resolución de X + 5 = B – C no se puede escribir LET X + 5 = = B – C que no tiene sentido en Basic, sino de la forma LET X = B – C –5.

Como el ordenador no puede leer más que una secuencia lineal de caracteres, no puede pues comprender expresiones como la:

$$\frac{A + B}{C + D} \qquad \text{o bien } X^2$$

que emplean varias líneas. Estas expresiones deben ser transcritas a otra forma empleando signos operativos distintos. Los signos son + para la adición, – para la sustracción, * para la multiplicación, / para la división y ↑ para la elevación de potencias.

$$X = \frac{A + B}{C - D} \qquad \text{debe pues escribirse}$$

```
100  LET X = (A + B)/(C – D)
```

y

$$Y = X^2$$

```
110  LET Y = X ↑ 2
```

Una operación debe siempre indicarse con un signo operativo. Mientras que en álgebra AB se entiende como el producto de A por B aunque no esté presente el signo "x", la instrucción LET C = AB se considera errónea por el ordenador y se debe escribir LET C = A * B.

Otra fuente de error en la escritura es la omisión de paréntesis, especialmente cuando se trata de fracciones:

La ecuación:

$$X = \frac{CD}{K + 0.5} \qquad \text{se debe escribir en BASIC}$$

```
LET X = C * D/(K + 0.5)
```

y no así

```
LET X = C * D/K + 0.5
```

que representa la ecuación $X = \frac{CD}{K} + 0.5$

En el caso que se tenga una división, el numerador y el denominador deben encontrarse entre paréntesis si contienen alguna suma o resta.

Para saber dónde son necesarios los paréntesis, conviene recordar que el ordenador en el cálculo de una expresión procede así:

1. En ausencia de paréntesis, el ordenador calcula primeramente:
   - las potencias,
   - las multiplicaciones y divisiones comenzando por la izquierda de la expresión,
   - las adiciones y sustracciones comenzando por la izquierda de la expresión.

En 10 LET X = A + E/F + B ↑ 2 + C/D * G,

- B ↑ 2 se calculará primero, después
- E/F, después
- C/D, después
- (C/D) * G, después
- (A + E/F), después
- (A + E/F) + B ↑ 2

después ((A + E/F) + B ↑ 2) + (C/D) * G. El resultado será pues $X = A + \frac{E}{F} + B^2 + \frac{C}{D} \times G$.

2. Cuando una expresión contiene paréntesis, el ordenador calcula primeramente el contenido de los paréntesis más interiores, comenzando por la izquierda de la expresión y empleando las reglas recién mencionadas:

En 10 LET X = (A + E)/(F + B ↑ (2 + C))/(D * G),

- 2 + C se calculará primero, después
- B ↑ (2 + C), después
- A + E, después
- F + (B ↑ (2 + C)), después
- D * G, después
- (A + E)/(F + (B ↑ (2 + C))), después
- ((A + E)/(F + (B ↑ (2 + C))))/(D*G).

El resultado será pues

$$\frac{A + E}{F + B^{(2+C)}} (D \times G)$$

que es evidentemente muy distinta de la expresión calculada sin paréntesis.

El empleo de paréntesis, aunque no sean necesarios, permite con frecuencia hacer una expresión más comprensible al programador y evitar algunos errores. No se debe dudar de emplear paréntesis aunque no se esté seguro de su utilidad.

Por ejemplo:

$$X = \frac{A + B^{2.54}.C}{B + \frac{AC}{B}} \qquad (C + Y)$$

puede escribirse

```
LET X = (A + B ↑ 2.54 * C)/(B + A * C/B) * (C + Y)
```

o bien

```
LET X = ((A + (B ↑ 2.54) * C)/(B + (A * C)/B)) * (C + Y)
```

y también de muchas otras formas.

Cuando el número de paréntesis en una expresión es elevado, es conveniente verificar que el número de paréntesis abiertos "(" es igual al de paréntesis cerrados ")".

EJERCICIOS

1. Entre las instrucciones siguientes indicar las que son incorrectas sintácticamente:

```
120  LET X = A + B/(X ↑.2 + K)
 60  LET Y = (A + C/(B/(A + 2.5)) + X ↑ 2.5
 10  LET T + S = A + B ↑ 2 − C
250  A = (A + B) * (C + D)/(K * L + 5)
  5  LET K = (X + Y) (A + B)
 18  LET A = − X * − Y
 80  LET M = X ↑ 2 + Y ↑ 2 + N ↑ − L
```

2. Escribir las instrucciones LET que permitan calcular el valor de X en las ecuaciones siguientes:

a $A = \dfrac{X + Y^2}{B}$ *b* $X - M = L(A + B)$

c $X - K = \dfrac{A^2 \times B^2}{C(L + M) - K}$ *d* $X = M + \dfrac{N}{A} + B$

SOLUCIONES

1. Las instrucciones siguientes son erróneas:

línea 60: número de paréntesis abiertos "(" diferente del número de paréntesis cerrados ")",
" 10: más de una variable a la izquierda del signo igual,
" 250: falta el LET
" 5: falta un signo operativo,
" 18: se deben separar dos signos operativos
18 LET A = - X * (-Y) es correcta,
" 80: se deben separar dos signos operativos
80 LET M = X ↑ 2 + Y ↑ 2 + N ↑ (- L) es correcta.

2. Las instrucciones siguientes permiten calcular el valor de X

a) 10 LET X = A * B - Y ↑ 2
b) 10 LET X = L * (A + B) + M
c) 10 LET X = A ↑ 2 * B ↑ 2/(C * (L + M) - K) + K
d) 10 LET X = M + N/A + B

4. LAS TRANSFERENCIAS DE CONTROL

Las instrucciones de un programa se ejecutan sencuencialmente en el orden en que se han escrito. Las instrucciones que permiten romper esta secuencia se llaman instrucciones de transferencia de control, y son de dos tipos: incondicionales y condicionales.

a) Transferencia de control incondicional

Se realiza con la instrucción GØ TØ que ya hemos encontrado antes. Esta instrucción debe ir seguida del número de la línea que contiene la próxima instrucción que el ordenador debe ejecutar.

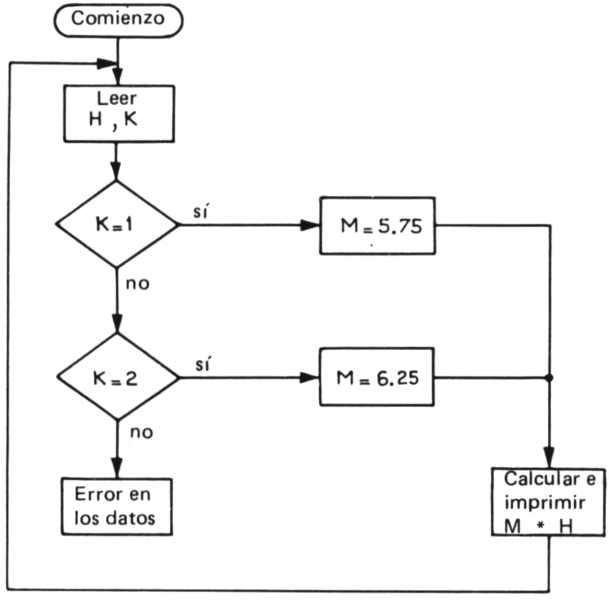
100 GØ TØ 50 es equivalente a la orden siguiente dada al ordenador: "ir a la línea 50, ejecutar la orden que contiene y seguir secuencialmente a partir de esta línea". Es una especie de vía fija que hace pasar el control a la línea 50 cada vez que se alcance la línea 100, de lo que se deriva el apelativo de incondicional.

b) Transferencia de control condicional

En el programa FACTURACION, se debe introducir cada vez el coste horario y la duración de la reparación. El coste horario es sin embargo fijo para cada obrero. Supongamos que se tengan sólo dos tipos de obreros. Se puede evitar el tener que indicar cada vez el coste horario del obrero sustituyéndolo por un código sencillo que se corresponda con él. Tomemos por ejemplo:

– Código 1 para un coste horario de 5,75 $,
– Código 2 para un coste horario de 6,25 $.

El cálculo del coste de la mano de obra se puede representar con el organigrama siguiente:



K es el código indicando la categoría del obrero. Si K = 1, se asigna a M el valor 5,75; pero si K = 2, el valor es 6,25. Cuando M ha sido determinado, se puede calcular el coste de la mano de obra M * H e imprimirlo. Se vuelve enseguida a leer un par de valores K y H y se rehacen los cálculos. El ordenador se para cuando no haya más datos. K puede tomar sólo los valores 1 y 2 ya que no hay más que dos categorías de obreros. Si se encuentra que K tiene otro valor, se ha cometido un error en los datos, en cuyo caso se imprime un mensaje indicando el error.

En dos puntos del programa, el ordenador deberá probar el valor de K y actuar de una forma condicional según el resultado de esta prueba, lo que se realiza con la instrucción de transferencia condicional IF THEN.

La instrucción 110 IF K = 1 THEN 150 es equivalente a la orden siguiente dada al ordenador "si el valor de K es 1, ir a la línea 150, ejecutar la orden que contiene y seguir secuencialmente a partir de esta línea.

Si el valor de K es diferente de 1, seguir normalmente a la línea siguiente (120)".

```
100  READ K,H
110  IF K = 1 THEN 150
120  IF K = 2 THEN 170
130  PRINT "ERROR EN LOS DATOS"
140  STØP
150  LET M = 5.75
160  GØ TØ 180
170  LET M = 6.25
180  PRINT M * H
190  GØ TØ 100
200  DATA 1,3,2,5,3,3,1,2,2,4
210  END
```

Para comprender este programa, sigamos paso a paso su ejecución con los datos de la línea 200. El ordenador lee primeramente los valores de K y H que son 1 y 3. La línea 110 pasa el control a la línea 150 ya que K = 1. M toma el valor 5,75 correspondiente al coste horario de los obreros de la categoría 1. La línea 160 hace pasar el control a la línea 180 que imprime el coste de la mano de obra M * H. La instrucción de la línea 190 permite repetir el cálculo varias veces. El lector puede seguir la ejecución con los pares de datos siguientes y ver cuándo se ejecutan las líneas 120 y 130. ¿Qué pasará cuando K = 3?

En la línea 140 se introduce una nueva instrucción: STOP que es muy explícita: cuando la encuentra en ejecución, el ordenador abandona este programa.

En este punto se pueden resumir rápidamente las diversas formas de parar la ejecución de un programa:

- La instrucción STØP,
- La instrucción END, que sirve para indicar el fin físico del programa (útil en la compilación) pero si el ordenador la encuentra durante la ejecución, tiene el mismo efecto que la instrucción STØP,
- La instrucción READ en un bucle lleva al final de la ejecución por agotamiento de la lista de datos.

Pero volvamos a la instrucción IF. Su formato general es IF (condición) THEN (número de línea). Si la condición se cumple, el control pasa a la línea cuyo número se encuentra después del THEN. Si no, se ejecuta la instrucción que va inmediatamente después de la instrucción IF.

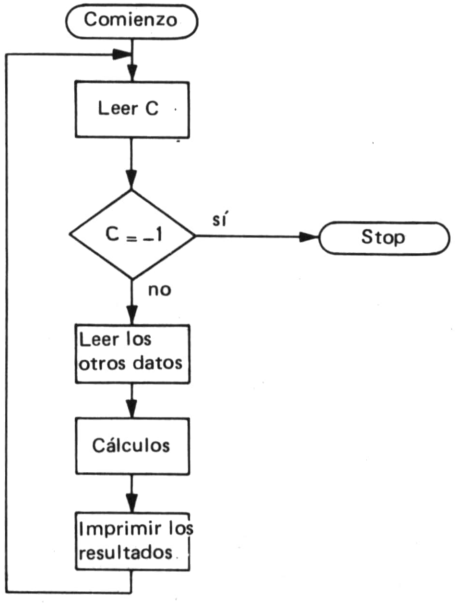
La condición es una comparación de dos *expresiones*. En el caso más sencillo, uno de los elementos de la comparación es una variable y el otro una constante, como sucede en nuestro ejemplo. Las comparaciones posibles en Basic son:

=	igual
>	mayor que
<	menor que
> =	mayor o igual que
< =	menor o igual que
< >	diferente a

Las instrucciones siguientes son correctas:

```
50  IF (X↑2 + 4.2) > = 2 * Q/(X + Y) THEN 100
50  IF X < > 10 THEN 90
50  IF 4 < (X * Q)↑2 THEN 40
50  IF X + A = - 3.24 THEN 523
```

Se puede por ejemplo emplear la instrucción IF para detener el programa FACTURAS. Para esto, se termina la lista de datos con un número peculiar: - 1 por ejemplo. Cada vez que se lea un número nuevo de cliente C, se prueba o mira su valor. Si es - 1 se salta a la instrucción END, lo que acaba la ejecución. Esto se ve mejor con el organigrama siguiente:



Se obtiene esto sustituyendo la instrucción 130 READ C, N, P, H, M, T por las instrucciones siguientes en el programa de la página 44:

```
130  READ C
134  IF C = - 1 THEN 340
138  READ N,P,H,M,T
```

Este método de parar la ejecución de un programa es el preferido normalmente. El mensaje "OUT OF DATA" puede indicar que realmente se ha cometido un error en los datos, y que no se han puesto los suficientes para realizar los cálculos. Su empleo para detener la ejecución de un programa puede siempre hacer creer en un fin normal, mientras que en realidad se ha hecho una lista de datos insuficiente.

EJERCICIOS

1. Se modifica el programa de la página 54 introduciendo la línea 125 y suprimiendo la línea 160 como sigue:

```
100  READ K, H
110  IF K = 1    THEN 150
120  IF K = 2    THEN 170
125  IF K < > 2 THEN 130
130  PRINT "ERROR EN LOS DATOS"
140  STØP
150  LET M = 5.75
170  LET M = 6.25
180  PRINT M * H
190  GØ TØ 100
200  DATA 1,3,2,5,3,3,1,2,2,4
210  END
```

   ¿Para qué sirve la línea 125? Los resultados dados por este programa ¿son siempre correctos? Si no es así ¿en qué caso son incorrectos?

2. ¿Son válidas las instrucciones siguientes?

a)
```
100  IF 15 = 6 THEN 120
```
b)
```
120  IF 0 <A <10 THEN 400
```
c)
```
(130  LET X = 180)
140  GØ TØ X
```
d)
```
150  IF X + 2 = Y ↑ 3 THEN 350
```
e)
```
160  IF A >B THEN GØ TØ 200
```
f)
```
170  IF Y + 1.5 >6 * X THEN A = B
```

3. Hacer un programa que lea el valor de la variable A e imprima 1 si $0 \leqslant A \leqslant 10$ y 2 si A es exterior a este intervalo.

SOLUCIONES

1. La línea 125 es inútil, ya que si se llega a ella es porque K es distinto de 2, pues si no se habría ido a la línea 170. Cualquiera que sea el valor de K, M siempre valdrá 6.25 ya que la instrucción de la línea 170 se ejecutará en ambos casos K = 1 y K = 2. Los resultados no serán exactos más que para K = 2, es decir, para la categoría 2 de obreros.

2. Las instrucciones siguientes son incorrectas:

   b) Sólo se puede poner una condición en una instrucción IF (y aquí hay 2: A > 0 y A < 10),

   c) GØ TØ tiene que ir seguido por un número de línea (en claro y no por una variable),

   e) THEN no puede ir seguido más que por un número de línea (y no por una instrucción GØ TØ),

   f) THEN no puede ir seguido más que por un número de línea (y no por una ecuación).

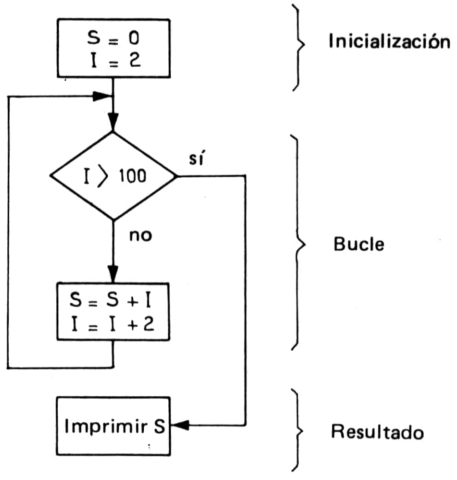
La instrucción a) es sintácticamente correcta. El control pasará siempre a la línea 120, por lo que es más sencillo y elegante escribir 100 GØ TØ 120.

3. El programa es:

```
100  INPUT A
110  IF A >10 THEN 150
120  IF A <0  THEN 150
130  PRINT "1"
140  STØP
150  PRINT "2"
160  END
```

III. BUCLES FOR

Supongamos que se quiera escribir un programa para sumar los números pares de 2 a 100 (2 + 4 + 6 + 8 + ... + 100). Un medio de realizar esto se indica con el organigrama siguiente:



Se definen dos variables:

S (con valor inicial 0) e I (con valor inicial 2). La adición se hace pasando 50 veces a través del bucle. Los valores de S e I se modifican como sigue:

Inicialización $\begin{cases} S = 0 \\ I = 2 \end{cases}$

1ª pasada $\begin{cases} S = S + I = 0 + 2 = 2 \\ I = I + 2 = 2 + 2 = 4 \end{cases}$

2ª pasada $\begin{cases} S = S + I = 2 + 4 = 6 \\ I = I + 2 = 4 + 2 = 6 \end{cases}$

3ª pasada $\begin{cases} S = S + I = 6 + 6 = 12 \\ I = I + 2 = 6 + 2 = 8 \end{cases}$

$\vdots$

49ª pasada $\begin{cases} S = S + I = 2352 + 98 = 2450 \\ I = I + 2 = 98 + 2 = 100 \end{cases}$

50ª pasada $\begin{cases} S = S + I = 2450 + 100 = 2550 \\ I = I + 2 = 100 + 2 = 102 \end{cases}$

Después de la 50ª pasada, S contiene la suma de los números pares de 2 a 100. Esto puede realizarse con el programa siguiente:

```
100  LET S = 0
110  LET I = 2
120  IF I > 100 THEN 160
130  LET S = S + I
140  LET I = I + 2
150  GØ TØ 120
160  PRINT S
170  END
```

La variable I controla el bucle, denominándose índice. Las magnitudes necesarias para controlar el bucle son: el valor inicial, el incremento de variación y el valor final de su índice. El valor inicial (2) se especifica en la línea 110. La línea 140 da el incremento de variación (+ 2) y el valor final (100) se compara con I en la línea 120.

El lenguaje BASIC permite realizar bucles de una forma más sencilla empleando la instrucción FØR. El programa anterior se puede escribir también así:

```
100  LET S = 0
110  FØR I = 2 TØ 100 STEP 2
120     LET S = S + I
130  NEXT I
140  PRINT S
150  END
```

La instrucción FØR I = 2 TØ 100 STEP 2 es una orden dada al ordenador equivalente a "ejecutar la porción de programa comprendida entre esta línea y la próxima instrucción NEXT I, de una forma sucesiva para valores del índice I creciendo de 2 a 100 con incrementos de 2"

El bucle está ahora definido por dos líneas:

- La línea 110 determina el valor inicial (I = 2), el valor final (TØ 100) y el incremento de variación (STEP 2) del índice I;
- La línea 130 determina el fin del bucle (cuyo índice es I).

Observemos que:

– Cada instrucción FØR debe ir seguida en el programa por una instrucción NEXT utilizando el mismo índice.

– Puede haber un mínimo ilimitado de instrucciones entre un FØR y el NEXT correspondiente,

– Una instrucción FØR no es necesaria para realizar un programa. Puede ser sustituída por otras instrucciones.

– Sin embargo, es más fácil construir un bucle con una instrucción FØR.

De una manera general, una instrucción FØR se escribe así:

FØR (nombre del índice) = (valor inicial)

TØ (valor final) STEP (incremento de variación)

Cuando el ordenador encuentra esta instrucción, da al índice el valor inicial indicado. Luego ejecuta las instrucciones siguientes hasta el NEXT correspondiente. Entonces añade el incremento de variación al valor actual del índice, probando el nuevo valor: si excede el valor final, ejecuta las instrucciones que están después del NEXT; pero si no, ejecuta de nuevo las instrucciones en el interior del bucle.

El valor inicial, el valor final y el incremento de variación pueden ser expresiones con valores enteros o reales positivos o negativos. Las instrucciones siguientes son válidas:

```
100  FØR I = 4.2 TØ (X * A)↑2 STEP B
 55  FØR A = K + B TØ N STEP 10
140  FØR Z = 50 TØ - 10 STEP - 5
2432 FØR L = - 50 TØ - 125 STEP - (A + B)↑2
```

Sin embargo, conviene comprobar que estas instrucciones se pueden ejecutar. Si se escribe por ejemplo:

```
100  FØR I = 50 TØ 10 STEP 20
 50  FØR I = 1  TØ 10 STEP - 1
```

o bien,

estas instrucciones no pueden ejecutarse ya que el índice de entrada excede ya al valor final. En este caso, el ordenador salta directamente a la primera línea que sigue al bucle FOR NEXT.

Cuando el incremento de variación es de 1, se puede omitir:

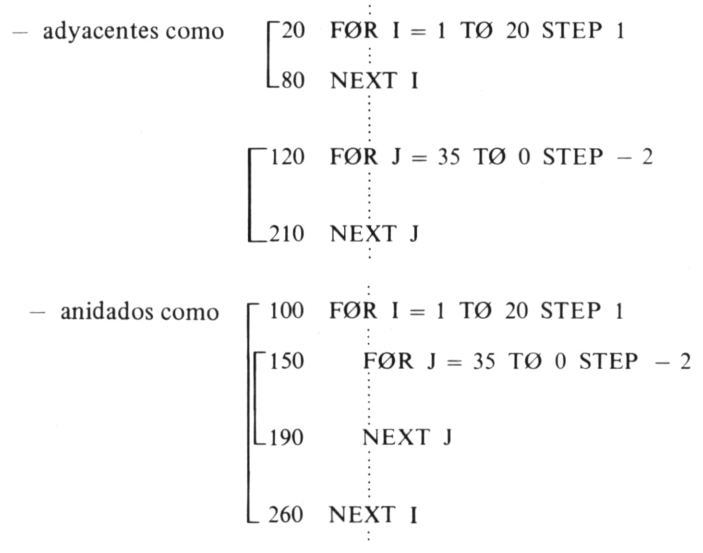
```
100 FØR I = 1 TØ 100
```

que equivale a

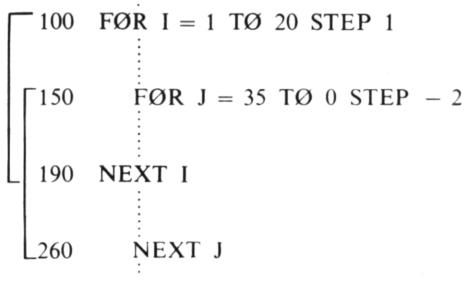
```
100 FØR I = 1 TØ 100 STEP 1
```

La repetición del nombre del índice después de la instrucción NEXT como en NEXT I es necesaria para indicar al ordenador a qué FØR se

refiere el NEXT. Esto es útil, pues se pueden tener en un programa varios bucles:



No es posible tener bucles que se superpongan parcialmente como:



lo que será considerado como un error por el ordenador.

Se aconseja, para leer los programas más fácilmente, escribir el contenido de los bucles interiores desplazando hacia la derecha con relación al contenido de los bucles exteriores, como se ha indicado arriba.

Uno de los papeles del índice es controlar el bucle; pero puede emplearse dentro del bucle en el cálculo de una expresión o en una instrucción IF por ejemplo. Pero conviene asegurarse de que no se modifica

por inadvertencia el valor del índice dentro del bucle, como en el ejemplo siguiente:

```
100  FØR I = 1 TØ 10
110     LET K = I
120     FØR I = 1 TØ 3
130          PRINT K ↑ I,
140     NEXT I
150     PRINT
160  NEXT I
170  END
```

Se ha escrito este programa para imprimir sobre una misma línea un número entero, su cuadrado y su cubo, y esto para los enteros de 1 a 10. El resultado será completamente distinto del deseado, puesto que el índice I del bucle 100 a 160 se usará igualmente en el bucle 120 a 140 y se modificará por tanto. Esto se corrige fácilmente sustituyendo I por otra variable en las líneas 120 a 140:

```
120  FØR L = 1 TØ 3
130       PRINT K ↑ L
140  NEXT L
```

Se puede salir de un bucle antes que el índice haya alcanzado su valor final. Basta para ello usar una instrucción que permita una transferencia de control (IF ó GØ TØ). Sin embargo, es más peligroso el entrar directamente en medio de un bucle sin pasar por la instrucción FØR como en el caso siguiente:

```
   :
 60  GØ TØ 150
   :
100  FØR I = 1 TØ 10 STEP 2
   :
150
   :
200  NEXT I
   :
```

En efecto, el valor de I se inicializa a 1 cuando el ordenador encuentra la instrucción FØR de la línea 100. Si se entra en el bucle por la instrucción de la línea 60, el ordenador no habrá pasado por la línea 100, y la I no se habrá inicializado y su valor será difícil de anticipar, lo que causará errores.

EJERCICIOS

1. Indicar dos errores en la secuencia de instrucciones siguiente:

```
100  LET X = 100
110  LET Y = 2
120  FØR  I = Y TØ 10 STEP 3
130              FØR J = X TØ I
140                   LET L = L + J
150              NEXT I
160              LET K = K + L
170              LET L = 0
180  NEXT J
190  PRINT K
200  END
```

2. ¿Qué hace el programa siguiente?

```
 90  LET S = 0
100  FØR I = 1 TØ 20 STEP 2
110      LET S = S + I↑2
120  NEXT I
130  PRINT S
140  END
```

3. Hacer un programa que permita hallar la parte entera de la raíz cuadrada de un número N.

4. Escribir el programa siguiente sin la instrucción FØR:

```
 90  LET X = 0
100  INPUT A,B
110  FØR I = A/2 TØ B↑2 STEP 10
120       LET X = X + I/4
130       IF X > 100 THEN 150
140  NEXT I
150  PRINT X
160  END
```

5. El programa siguiente no da siempre los mismos resultados que el anterior. ¿En qué caso difieren los resultados de estos dos programas?

```
 90  LET X = 0
100  INPUT A,B
110  LET I = A/2
120  LET X = X + I/4
130  IF X >100 THEN 160
140  LET I = I + 10
150  IF I <= B↑2 THEN 120
160  PRINT X
170  END
```

6. ¿Qué hace el programa no corregido de la página 62?

SOLUCIONES

1. Los bucles FØR no deben estar ahorquillados. El NEXT I (línea 150) debería hallarse después del NEXT J (línea 180).

   La instrucción de la línea 130 no es ejecutable: el valor inicial X (100) es superior al valor final (I = Y = 2) al comienzo, siendo su incremento positivo (1).

2. El programa calcula e imprime la suma de cuadrados de los números impares de 1 a 19.

3.
```
100  READ N
110  FØR I = 1 TØ N
120          IF N < I↑2 THEN 140
130  NEXT I
140  PRINT I - 1
150  DATA 2432
160  END
```

4.
```
 90  LET X = 0
100  INPUT A,B
110  LET I = A/2
120  IF I >B↑2 THEN 170
130  LET X = X + I/4
140  IF X >100 THEN 170
150  LET I = I + 10
160  GØ TØ 120
170  PRINT X
180  END
```

5. Los programas de las preguntas (4) y (5) dan resultados diferentes cuando A/2 es superior a B^2. En este caso, el ordenador no ejecutará las instrucciones del bucle FØR y saltará directamente a la línea 150 del programa (4) para imprimir el valor de X que es siempre 0. Al contrario, en el programa (5) X se calcula antes de que se haga la prueba con el valor de I. El valor de X impreso será diferente de 0 e igual a A/8.

6. Sigamos paso a paso lo que hace el programa:

Número de la instrucción ejecutada	
100	I = 1
110	K = I = 1
130	$K^1 = 1^1 = 1$ se imprime
140	I = I + 1 = 2
130	$K^1 = 1^2$ se imprime en la misma línea
140	I = I + 1 = 3
130	$K^1 = 1^3$ se imprime en la misma línea
140	I es igual a 3. El control pasa a la línea 150
150	El carro del teletipo avanza un paso
160	I = I + 1 = 4

```
110  K = I = 4
120  I = 1
130  K^1 = 4^1 = 4 se imprime
140  I = I + 1 = 2
130  K^1 + 4^2 = 16  se imprime
140  I = I + 1 = 3
130  K^1 = 4^3 = 64  se  imprime
140  I es igual a 3. El control pasa a la línea 150
150  El carro del teletipo avanza un paso
160  I = I + 1 = 4
110  K = I = 4
```

El ciclo se repite indefinidamente hasta que el usuario detenga manualmente el ordenador. La hoja de resultados obtenida es:

1	1	1
4	16	64
4	16	64
4	16	64

Algunos sistemas detectarán sin embargo, el error y la ejecución del programa no podrá comenzar.

IV. LISTAS Y TABLAS

Para almacenar una magnitud en el ordenador, basta definir una variable a la que se le da un nombre. Pero a veces se tiene un cierto número de variables que representan magnitudes de la misma naturaleza. Consideremos el almacén de un taller que tenga cinco clases de piezas de recambio para un determinado tipo de máquina. Se pueden escoger las variables A, B, C, D y E para representar el precio de las piezas de categorías respectivas 1, 2, 3, 4 y 5. Para acordarnos de que las cinco variables representan cantidades de la misma naturaleza, es preferible denominarlas P1, P2, P3, P4 y P5.

Esta notación es una buena regla mnemotécnica; pero no aporta, sin embargo, ninguna simplificación en la escritura de un programa. Supongamos por ejemplo que se llamen N1, N2, N3, N4 y N5 los números respectivos de piezas en existencia para las categorías 1, 2, 3, 4 y 5. Se

quieren calcular los valores de inventario V1, V2, V3, V4 y V5 para cada categoría de piezas. El programa siguiente permite hacer este cálculo:

```
100  READ P1, P2, P3, P4, P5
110  READ N1, N2, N3, N4, N5
120  LET V1 = P1 * N1
130  LET V2 = P2 * N2
140  LET V3 = P3 * N3
150  LET V4 = P4 * N4
160  LET V5 = P5 * N5
170  DATA 1.2, 2.5, 8, 0.9, 4.75
180  DATA 5,1,4,0,3
190  END
```

Las líneas 120 a 160 hacen el mismo cálculo para las piezas de las diferentes categorías. Si el almacén tuviera 100 clases de piezas, se deberían escribir 100 líneas para hacer estos cálculos.

Sería evidentemente más sencillo tener un grupo de instrucciones que indiquen sencillamente al ordenador "para cada categoría de piezas, calcular el valor de inventario multiplicando el número de piezas por el precio unitario correspondiente a esta categoría".

Esto puede realizarse gracias a las *listas*. La lista de los precios se escribe: P (1), P (2), P (3), P (4), P (5). Cada elemento de una lista está definido por:

- El nombre de la lista a la que pertenece: P
- Su número de orden en la lista o *índice*. P (3) es el tercer elemento de la lista P. El índice debe ir entre paréntesis y siguiendo al nombre de la lista. Por esto se llama a veces a una lista una *variable indexada*.

Las listas para el número de piezas de recambio de cada categoría y el valor de los diferentes inventarios se pueden definir así:

```
N(1), N(2), N(3), N(4), y N(5)
V(1), V(2), V(3), V(4), y V(5)
```

El programa anterior se puede escribir más sencillamente empleando estas listas:

```
100  FØR I = 1 TØ 5
110        READ P(I), N(I)
120        LET V(I) = P(I) * N(I)
130  NEXT I
140  DATA 1.2, 5, 2.5, 1,8,4,0.9,0,4.75, 3
150  END
```

En el interior del bucle FØR-NEXT, los elementos de la lista tienen a I como índice. Primeramente I toma el valor 1. La instrucción 110 lee los valores de P (1) y N (1) y luego la instrucción 120 calcula V (1). Enseguida I toma el valor 2. Se leen P (2) y N (2), luego se calcula V (2) y así sucesivamente hasta que se hayan calculado los valores de los cinco inventarios. Si el almacén tuviera 100 clases de piezas, bastaría cambiar la línea 100 por ésta: 100 FØR I = 1 TØ 100 y completar la lista de datos.

La utilización de la instrucción FØR y de las listas, permite tratar más fácilmente un gran número de variables que representen una misma magnitud. Las listas aportan igualmente una flexibilidad mayor en la notación de las variables. En efecto, mientras que el número máximo de variables no indexadas es 286, no hay prácticamente ningún límite en el número de elementos que pertenezcan a listas (recordemos que una variable no indexada debe tener como nombre una letra o bien una letra seguida de una cifra. Esto da 11 posibilidades para cada letra, por ejemplo A, A0, A1, A2, ..., A9. O sea un máximo de 26 x 11 = 286 variables no indexadas).

En nuestro ejemplo, se ha supuesto para simplificar que había cinco categorías de piezas. Supongamos que estas cinco categorías existen para cada uno de los dos tipos de máquinas usadas en el taller. Es poco probable que una pieza de una cierta categoría cueste lo mismo para cada tipo de máquina. El precio unitario de una pieza depende pues de dos criterios: la categoría de la pieza y el tipo de la máquina. ¿Cómo se pueden representar las variables precio, número de piezas y valor del inventario ahora que dependen de dos criterios?

Una solución consiste en definir las listas teniendo un nombre diferente para cada tipo de máquina. Se llamará respectivamente:

P (I) y Q (I) a los precios unitarios de las piezas de los tipos 1 y 2; M (I) y N(I) su número y V (I) y W (I) el valor de los inventarios, correspondiendo I al número de la categoría.

P (3) será así el precio unitario de una pieza de la categoría 3 para el tipo 1 de máquina. W (5) será el valor del inventario de las piezas de la categoría 5 para el tipo 2 de máquina.

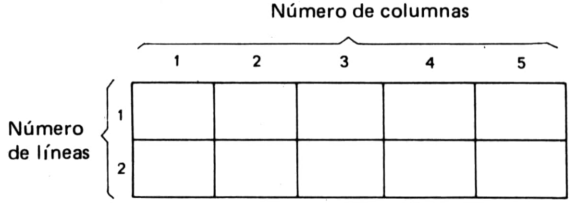
El programa que permite calcular el valor de los diferentes inventarios se escribe:

```
100  FØR I = 1 TØ 5
110        READ P (I), Q (I), M (I), N (I)
120        LET V (I) = P (I) * M (I)
130        LET W (I) = Q (I) * N (I)
```

```
140  NEXT I
150  DATA 1.2,5, 2.5, 1,8,4, 0.9,0,4.75, 3
160  DATA 3, 4, 8.3, 0, 9.5, 2, 1.5, 4, 6, 1
170  END
```

Las líneas 120 y 130 hacen el mismo cálculo para los dos tipos de máquinas; si el taller tuviera 50 tipos de máquinas se deberían escribir 50 líneas para hacer los cálculos. Sería mucho más simple tener un grupo de instrucciones que indiquen sencillamente al ordenador "Para cada categoría de piezas y para cada tipo de máquinas, calcular el valor del inventario multiplicando el número de piezas por el precio unitario correspondiente".

Esto puede realizarse gracias a las *tablas*. Una tabla de 2 líneas y 5 columnas puede representarse como sigue:



Se puede por ejemplo crear una tabla de precios P. Cada elemento P (I, J) de esta tabla está definido por:

- El nombre de la tabla a la que pertenece P,
- Su número de línea I que corresponde aquí al tipo de máquina,
- Su número de columna J que corresponde aquí a la categoría de la pieza.

P (2,4) es el elemento de la tabla de precios situado en la segunda línea y en la cuarta columna. Representa el precio de una pieza de la categoría 4 para el tipo 2 de máquina.

Se define de la misma forma una tabla para los números de pieza en cada inventario N (I, J) y una tabla para los valores de cada inventario V (I, J).

El programa de la página anterior puede escribirse ahora:

```
100  FØR I = 1 TØ 2
110        FØR J = 1 TØ 5
120              READ P (I,J), N (I,J)
130              LET V (I,J) = P (I,J) * N (I,J)
140        NEXT J
150  NEXT I
160  DATA 1.2, 5, 2.5, 1, 8, 4, 0.9, 0, 4.75, 3
170  DATA 3, 4, 8.3, 0, 9.5, 2, 1.5, 4, 6, 1
180  END
```

El bucle FØR-NEXT exterior lleva el índice I representando el tipo de máquina. El bucle interior lleva el índice J representando la categoría de las piezas. Al comienzo I vale 1. J va variando de 1 a 5 y se calculan los valores de los inventarios V (1,1), V (1,2), V (1,3), V (1,4) y V (1,5) correspondientes a las 5 categorías de piezas para la máquina 1 en la línea 130. Luego I toma el valor 2 y ahora se calcularán V (2,1), V (2,2), V (2,3), V (2,4) y V (2,5). Para tener enseguida por ejemplo el valor del inventario de piezas de la categoría 4 para el tipo 2 de máquinas, bastará consultar el valor de V (2,4).

La parte cálculo de este programa contiene 6 instrucciones. El programa precedente tenía sólo 5. ¿Por qué complicar la cosa para escribir un programa más largo? De hecho, sería raro que el número de los tipos de máquinas en el taller fuera solamente igual a 2. Si son 50, basta cambiar la línea 100 para escribir 100 FØR I = 1 TØ 50.

El número de instrucciones es siempre igual a 6, mientras que en el programa anterior se habrían necesitado 50 instrucciones LET teniendo la misma estructura que la de la línea 120. El número total de instrucciones sería de 53.

Los programas anteriores son sólo ejemplos de las nuevas posibilidades que ofrece la utilización de las listas y tablas. Para poder aprovechar plenamente los servicios que pueden rendir, es necesario precisar su empleo:

Las listas y tablas deben representarse con sólo una letra.

N (3), X (3,2), X (3,3), A (8,103) son correctos.

N1 (3), XA (3,2), A7 (8,103) no son aceptables en BASIC.

Para cada lista y tabla, se reservan automáticamente sus "celdas" en memoria para que los índices puedan variar hasta 10. Si ciertos índices toman valores superiores a 10, las listas y tablas correspondientes deben ser "declaradas". Es decir, que en alguna parte del programa se deben indicar las dimensiones máximas de estas listas y tablas para que el ordenador pueda reservarlas sitio en la memoria. Esto se realiza gracias a una

instrucción DIM (de DIMENSION) seguida del nombre de la lista o de la tabla con los valores máximos de sus índices. Por ejemplo, si la tabla A tiene un máximo de 30 líneas y 50 columnas en un programa, bastará declararla así:

```
130  DIM  A (30, 50)                    (1)
```

Estas instrucciones pueden encontrarse en cualquier parte del programa, pero *antes que se usen las tablas y listas correspondientes.* Sin embargo, para evitar errores se aconseja agruparlas al principio del programa. Una sola instrucción DIM puede servir para declarar varias listas y tablas:

```
100  DIM A (100)
110  DIM K (35,72)
120  DIM H (54)
```

puede escribirse 100 DIM A (100), K (35,72), H (54).

El mayor valor permitido para un índice es 1022. Los índices en una declaración DIM deben ser explícitos, es decir que deben ser representados sólo con números y no con variables o expresiones.

```
100  DIM  A (100)  es correcto
100  DIM  A (N)  y  100  DIM  B (K * J + I)  no lo son
```

Fuera de las instrucciones DIM, las listas y tablas pueden tener expresiones como índices:

```
120  LET A ((I + 3) * N/K) = B (M/4) + 10.5
130  IF  A (I/2 + N * 5, X ↑ 0.5) = 2  THEN  200
```

son sintácticamente correctas en BASIC. El ordenador calcula el valor de las expresiones y las redondea al entero más cercano para determinar el valor del índice. En ciertos sistemas, por el contrario se toma el mayor entero inferior al valor de la expresión considerada.

Las listas y tablas pueden emplearse en cualquier parte del programa, lo mismo que las variables no indexadas. La sola excepción es la instrucción FØR. El índice usado en una instrucción FØR no puede ser más que una variable no indexada.

```
100  FØR  I = A * K  TØ  (X ↑ 0.5) * L  STEP  L * 2
```

(1) En el sistema BASIC desarrollado en el Dartmouth College, esta instrucción habría reservado lugar para una tabla de 31 líneas y 51 columnas, ya que se autoriza el índice 0. La mayor parte de los sistemas actuales no permiten utilizar índices inferiores a 1.

es correcta, pero no lo es:

```
100 FØR A (I) = A * K TØ (X ↑ 0.5) * L STEP L * 2
```

EJERCICIOS

1. Encontrar por lo menos cuatro faltas de sintaxis en el programa siguiente:

```
100  INPUT N
110  DIM A (N)
120  FØR I = 1 TØ N
130       INPUT A (I)
140  NEXT I
150  FØR B (4) = 1 TØ N
160        LET C (I) = A(I) * B (4)
170        PRINT C (I)
180  NEXT I
190  DIM B (30), C (30)
200  END
```

2. ¿Qué hace el programa siguiente?

```
100  DIM A (30)
110  FØR I = 1 TØ 30
120          LET A (I) = I
130  NEXT I
140  PRINT A (4.2), A (3.8), A (30), A (30.4)
150  END
```

3. ¿Qué hace el programa siguiente?

```
 50  FØR I = 1 TØ 10
 60        FØR J = 1 TØ 10
 70             LET X (I,J) = J
 80        NEXT J
 90  NEXT I
100  FØR J = 1 TØ 10
110           PRINT X (J,4)
120  NEXT J
130  END
```

4. Hacer un programa que imprima y guarde en memoria los cuadrados y cubos de los 50 primeros números enteros.

SOLUCIONES

1. Las instrucciones siguientes son sintácticamente falsas:
   - 110 DIM A (N) los índices deben ser explícitos en una declaración DIM,
   - 150 FØR B (4) = 1 TØ N el índice en un bucle FØR no puede ser más que una variable *no indexada*,

- 180 NEXT I el índice que sigue a NEXT no se corresponde con el índice del FØR,
- 190 DIM B (30), C (30) las listas y tablas deben declararse antes de emplearse.

2. Este programa coloca los 30 primeros números enteros en la lista A:

$$A(1) = 1, \quad A(2) = 2, \quad A(3) = 3, \ldots, A(30) = 30$$

Los valores impresos son pues:

4, 4, 30, 30

3. La tabla X tiene 10 líneas y 10 columnas, por lo que no se ha declarado. Cada elemento de esta tabla es igual al número de la columna a que pertenece: todos los elementos de la columna 1 son iguales a 1, todos los elementos de la columna 2 son iguales a 2 ...

Los valores impresos son los elementos de la columna 4, es decir, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4

4.–

```
100 DIM A (50), B (50)
110 PRINT "I", "I2", "I3"
120 FØR I = 1 TØ 50
130         LET A (I) = I ↑ 2
140         LET B (I) = I ↑ 3
150         PRINT I, A(I), B(I)
160 NEXT I
170 END
```

La lista A contiene los cuadrados y la lista B los cubos de los 50 primeros números enteros. El ordenador imprime:

I	I2	I3
1	1	1
2	4	8
3	9	27
4	16	64
⋮	⋮	⋮
50	2 500	125 000

V. FUNCIONES Y SUBRUTINAS

Ciertos tipos de cálculo aparecen con frecuencia en la escritura de programas: por ejemplo el cálculo de líneas trigonométricas en los problemas científicos. O a veces un bloque de instrucciones debe repetirse varias veces en un mismo programa para hacer cálculos con diferentes

valores de los parámetros. En estos dos casos, la utilización de funciones y de subrutinas simplifican la tarea del programador.

Las funciones comprenden:

– Las funciones definidas por el sistema,

– Las funciones definidas por el usuario que se estudiarán aparte.

1. FUNCIONES DEFINIDAS POR EL SISTEMA

Revisemos el programa elemental de la página 33. Es evidente que el importe de una factura no se escribirá por ejemplo 1485, 4923 $, sino más bien 1485,49 $, es decir, con un número entero de céntimos.

Antes de imprimir este importe, será necesario transformarlo para eliminar las fracciones de céntimo. Esta transformación se puede realizar gracias al programa siguiente:

```
100  INPUT X
110  LET X = X * 100
120  LET K = 0.01 * INT (X)
130  PRINT K
140  GØ TØ 100
150  END
```

La expresión INT ( ) que aparece en la línea 120 se llama una *función*. La X en INT (X) se llama el *argumento* de la función INT. Esta función tiene un valor igual al mayor número entero inferior a su argumento.

Por ejemplo:

INT (9.864) es igual a 9

INT (- 1421.02) es igual a -1422

100 LET K = INT (9.864) asigna pues el valor 9 a la variable K. La instrucción de la línea 110 transforma el valor en dólares X en un valor en céntimos al multiplicarlo por 100. En la línea 120, se eliminan las fracciones de céntimo e INT (X) representa un número entero de céntimos, que se multiplica por 0,01 para dar a K su valor en dólares.

El argumento contenido en la función INT puede ser una expresión cualquiera. Es válido en BASIC: INT ((X * Y) / A ↑ 0.5). En este caso, el ordenador calcula primero el valor del argumento y después el de la función.

La función INT se dice "definida por el sistema" (o "incorporada en el lenguaje") ya que el ordenador comprende el significado de esta función sin que el usuario tenga que definirla. Hay 11 funciones definidas en el lenguaje BASIC:

SIN (X) = seno del ángulo X
COS (X) = coseno del ángulo X
TAN (X) = tangente del ángulo X
ATN (X) = ángulo que tiene una tangente igual a X
EXP (X) = exponencial de X ($= e^X$)
ABS (X) = valor absoluto de X
SQR (X) = raíz cuadrada de X
LOG (X) = logaritmo neperiano de X
INT (X) = el mayor entero inferior a X
SNG (X) = indica el signo de X = + 1 si X > 0
= 0 si X = 0
= - 1 si X < 0
RND (X) = un número aleatorio entre 0 y 1.

Las cuatro primeras funciones son las funciones trigonométricas clásicas, estando los ángulos definidos en radianes.

EXP (X) calcula el valor de la constante *e* (= 2,7182818...) a la potencia X:

100 LET = 2.71828 ↑ X es pues aproximadamente equivalente a:

100 LET K = EXP (X)

La función ABS es igual al valor absoluto de su argumento, es decir:

ABS (X) = X si X > 0
= − X si X < 0

Si X = - 15,45 la instrucción 100 LET Y = ABS (X) dará a Y el valor 15,45. La variable X por supuesto no habrá cambiado de valor y será siempre igual a -15,45.

El argumento de las funciones LØG (logaritmo) y SQR (raíz cuadrada) deberá ser positivo. Si en la ejecución de un programa el ordenador encuentra una de estas dos funciones con un argumento negativo, se imprime un mensaje de error y la función se calcula con el valor absoluto del argumento.

Por ejemplo:

```
100  LET A = - 0.5
110  LET B = 8
120  LET C = SQR (A * B)
130  PRINT C
140  END
```

tendrá como resultado:

```
SQUARE RØØT ØF A NEGATIVE NUMBER
2
```

Antes de ver la función RND que es bastante peculiar, se impone hacer dos observaciones para precisar el empleo de las funciones definidas por el sistema:

1. No deben aparecer nunca a la izquierda de un signo igual en una instrucción LET. La instrucción 100 LET SQR (X) = 2 hará que el programa no se ejecute.
2. Pueden tener como argumento una expresión que contenga otra función definida por el sistema. Por ejemplo la instrucción:

```
100  LET K = INT (CØS (A) ↑ 2 + B)
```

es sintácticamente correcta.

La función RND (1) se emplea sobre todo en programas de simulación. Entrega un número aleatorio comprendido entre 0 y 1 (pero nunca igual a 0 ó a 1). La función RND debe poseer un argumento, pero este argumento no tiene ninguna influencia sobre el valor de la función y puede escogerse arbitrariamente. La secuencia de instrucciones siguiente permite obtener una lista de 20 números aleatorios comprendidos entre 0 y 1.

```
100  FØR I = 1 TØ 20
110        PRINT RND (A)
120  NEXT I
130  END
```

(1) RND es la abreviatura de la palabra inglesa RANDOM (azar).

Una simulación de 20 tiradas de un dado se obtiene con el pequeño programa siguiente:

```
100  FØR I = 1 TØ 20
110      PRINT INT (6 * RND (A) + 1)
120  NEXT I
130  END
```

El número aleatorio entero comprendido entre 1 y 6 se genera en la línea 110 antes de imprimirse:

- 6 * RND (A) es un número aleatorio real entre 0 y 6 (sin comprender los extremos).
- 6 * RND (A) + 1 es un número aleatorio real entre 1 y 7 (sin comprender los extremos, es decir, desde 1,00001 a 6,99999 limitándose a 6 cifras significativas).
- INT (6 * RND (A) + 1) da pues un número aleatorio entero de 1 a 6 (extremos incluidos).

EJERCICIOS

1. ¿Qué hace el programa siguiente? Compararlo con el de la página 73.

```
100  INPUT X
110  LET K = 0.01 * INT (X * 100 + 0.5)
120  PRINT K
130  GØ TØ 100
140  END
```

2. ¿Qué hace el programa siguiente?

```
100  FØR I = 1 TØ 20
110      LET D1 = INT (6 * RND (X) + 1)
120      LET D2 = INT (6 * RND (X) + 1)
130      LET D3 = INT (6 * RND (X) + 1)
140      IF D1 + D2 + D3 < >7 THEN 180
150      IF D1 = 4 THEN 200
160      IF D2 = 4 THEN 200
170      IF D3 = 4 THEN 200
180      PRINT "PIERDE"
190      GØ TØ 210
200      PRINT "GANA"
210  NEXT I
220  END
```

SOLUCIONES

1. Este programa redondea un valor en dólares al *céntimo más cercano*. El programa de la página 73 redondea un valor en dólares al *céntimo inferior.*

2. Este programa simula el juego del 421 para 20 tiradas de dados. Las instrucciones 110 a 130 asignan 3 valores aleatorios de 1 a 6 a los dados D1, D2, D3. Si la suma de estos tres valores es diferente de 7 (4 + 2 + 1) no se tiene un 421 y el programa imprime "PIERDE" en la línea 180.

   Si por el contrario la suma anterior es 7 y uno de ellos es 4, se tiene un 421, ya que la suma de los otros dos es 3 dando las únicas combinaciones 1,2 y 2,1.

   Las líneas 150 a 170 prueban los valores D1, D2, D3: si uno de ellos es igual a 4, el programa imprime "GANA", en otro caso imprime "PIERDE"

   Las 20 veces de iteración corresponden a 20 tiradas de los dados

2. *FUNCIONES DEFINIDAS POR EL USUARIO*

El programa de la página 76 permite redondear un importe al céntimo más cercano. Si esta transformación debe hacerse en varios sitios de un programa, puede resultar incómodo el tener que escribir cada vez la expresión 0.01 * INT (X * 100 + 0.5) mientras que X es lo único que cambia. Sería más práctico tener una función dando el valor de esta expresión y teniendo sólo a X como argumento. Esto puede realizarse gracias a una *definición de función* de la manera siguiente:

```
100  DEF FNA (X) = 0.01 * INT (X * 100 + 0.5)
```

La instrucción DEF indica que lo que sigue es la definición de una función. Esta función tiene aquí el nombre FNA y por valor 0.01 * INT (X * 100 + 0.5)

El programa de la página 76 puede escribirse de nuevo usando esta nueva función:

```
100  DEF FNA (X) = 0.01 * INT (X * 100 + 0.5)
110  INPUT A
120  LET K = FNA (A)
130  PRINT K
140  GØ TØ 110
150  END
```

El nombre dado al argumento en la definición de la función no tiene ninguna importancia. Se podría también haber escrito:

```
100 DEF FNA (L)  = 0.01 * INT  (L * 100 + 0.5)
```

o bien

```
100 DEF FNA (Z4) = 0.01 * INT (Z4 * 100 + 0.5)
```

sin cambiar nada el resto del programa y obteniendo los mismos resultados. Existe sin embargo una restricción sobre la naturaleza del argumento en la definición de una función: éste debe ser una variable *no indexada.*

Cuando el ordenador encuentra una instrucción definida por el usuario, sigue la secuencia de órdenes siguiente:

1. Buscar en qué sitio ha sido definida esta función (aquí en la línea 100).
2. Dar al *argumento ficticio* (aquí X) la definición, el valor del argumento real (aquí A).
3. Calcular la función usando la expresión contenida en su definición (aquí 0.01 * INT) (X * 100 + 0.5).

Se puede decir que una instrucción DEF sirve para explicar al ordenador cómo calcular una función no definida por el lenguaje BASIC.

Las reglas para la definición y el empleo de funciones son:

1. Una función definida por el usuario debe tener un nombre de tres letras, comenzando por FN (abreviatura de FUNCION). Se pueden pues definir las funciones FNA, FNB ... FNZ o sea un máximo de 26 funciones en un mismo programa.
2. Una función puede definirse en cualquier punto del programa, antes o después de su uso.
3. La definición de una función puede contener una función definida por el sistema como era el caso para FNA en nuestro ejemplo. Pero ella no puede contener a otra función definida por el usuario.

   110 DEF FNB (Y) = Y * FNA (Y) no es aceptable en BASIC pues la definición de la función FNB usa otra función FNA definida por el usuario.
4. Una función definida por el usuario puede emplearse en un programa de la misma manera que una función definida por el sistema.

El empleo de estas funciones simplifica la tarea del programador cuando una misma expresión debe escribirse varias veces en un progra-

ma. Es necesario, sin embargo, evitar definir una función que se utilice menos de tres veces, ya que se disminuye la eficacia del programa sin aumentar su sencillez sustancialmente.

EJERCICIOS

1. Las instrucciones siguientes son independientes. ¿Cuáles son incorrectas? ¿Por qué?

```
100  DEF FNX (X) = X ↑ 0.5 + CØS (X)
110  LET K = Y * FN3 (A2)
120  LET Z2 = (X7 ↑ 3 + C) + FNG (3 * X + Y ↑ 2)
130  DEF FNP (L4) = L4 ↑ 3
140  DEF FNK (Y) = Y * FNA (Y)
```

2. ¿Qué hace el programa siguiente?

```
100  READ I
110  FØR N = 1 TØ 20
120          LET K = FNR (N)
130          PRINT N, K
140  NEXT N
150  DEF FNR (X) = (1 + I) ↑ X
160  DATA 0.10
170  END
```

3. Escribir definiciones de funciones que permitan el cálculo de las expresiones siguientes:

$$3.1416\ R^2,\ X^2 + X^3,\ 4\ X^2 + 3X + 5$$

SOLUCIONES

1. Las instrucciones siguientes son incorrectas:

   110 LET K = Y * FN3 (A2). El nombre de una función debe ser de *tres letras*, siendo las dos primeras FN.

   140 DEF FNK (Y) = Y * FNA (Y). Una función definida por el usuario no puede emplearse en la definición de otra función.

2. Este programa permite calcular una tabla de interés compuesto para una tasa del 10% y un período que varía de 1 a 20 años.

3. Las definiciones siguientes son correctas:

```
100  DEF FNA (X) = 3.1416 * R ↑ 2
110  DEF FNB (X) = X ↑ 2 + X ↑ 3
120  DEF FNC (X) = 4 * X ↑ 2 + 3 * X + 5
```

3. SUBRUTINAS

Hemos visto cómo la definición de una función por el usuario evitaba el tener que escribir varias veces un mismo cálculo en un programa. El empleo de una función es, sin embargo, muy limitado, pues no se aplica más que cuando se calcula un *valor único* a partir de un *argumento único* con ayuda de una *expresión única.*

Sucede, sin embargo, que un bloque de varias instrucciones aparezca varias veces en un programa. Consideremos por ejemplo el caso de un programa en el que un código (por ejemplo, un código de cliente), lo entra el usuario en diferentes sitios. Es preferible de probar un código después de haberlo leído para disminuir los riesgos de errores (debidos a una mala transmisión o a un fallo de tecleo). Si este código debe ser un *entero positivo inferior a 1000*, el programa puede tener el aspecto siguiente:

```
                ⋮
200  INPUT C
210  IF C < 1 THEN 240
220  IF C > 999 THEN 240
230  IF C = INT (C) THEN 260
240  PRINT "ERROR EN LOS DATOS. REPETIR"
250  GØ TØ 200
260             ⋮

350  INPUT B
360  IF B < 1 THEN 390
370  IF B > 999 THEN 390
380  IF B = INT (B) THEN 410
390  PRINT  "ERROR EN LOS DATOS. REPETIR"
400  GØ TØ 350
410             ⋮
```

Si este procedimiento de prueba se debe hacer 10 veces en el programa, hará falta tener 10 veces un conjunto de instrucciones teniendo la misma estructura que la de las líneas 200 a 250. Sería mucho más práctico escribir este procedimiento una vez sólo y poder referirse a él cuan-

do se quiere leer y verificar un código. Lo anterior puede realizarse empleando una subrutina de la manera siguiente:

```
                    .
                    .
200  GØ SUB 1000
210  LET C = A
                    .
                    .
                    .
310  GØ SUB 1000
320  LET B = A
                    .
                    .
1000 REM  SUBRUTINA
1010 INPUT A
1020 IF A < 1 THEN 1050
1030 IF A > 999 THEN 1050
1040 IF A = INT (A) THEN 1070
1050 PRINT "ERROR EN LOS DATOS.
        REPETIR"
1060 GØ TØ 1010
1070 RETURN
                    .
                    .
```

Subrutina

En este programa se introducen dos nuevas instrucciones:

GØ SUB y RETURN

La subrutina se encuentra en las líneas 1000 a 1070 y permite entrar y verificar una variable A. Si el valor de entrada no es correcto se imprime un mensaje y el usuario debe teclear otro valor.

La instrucción GØ SUB 1000 permite hacer la transferencia de control a la línea 1000 como una instrucción GØ TØ pero juega igualmente otro papel.

200 GØ SUB 1000 es equivalente a la orden siguiente dada al ordenador: "poner en memoria el número de la próxima línea (210) después ejecutar la instrucción contenida en la línea 1000". El ordenador ejecuta pues las instrucciones de las líneas 1000 a 1070 leyendo y verificando así la variable A. Cuando este cálculo ha terminado, encuentra la instrucción RETURN.

1070 RETURN es equivalente a la orden siguiente dada al ordenador "transferir el control a la línea cuyo número ha sido puesto en memoria cuando la última instrucción GØ SUB ha sido encontrada"

El número es 210, luego la próxima instrucción a ejecutarse es 210 LET C = A que asigna a C el valor de A ya leído y verificado.

El ordenador sigue ahora trabajando sobre el programa principal. Cuando se encuentra la línea 310, se transfiere el control a la línea 1000, se lee otro valor de A y se verifica por la subrutina, después se asigna a B en la línea 320 y se ejecuta el resto del programa. Esto pone en evidencia dos reglas fundamentales en el empleo de subrutinas:

- Una subrutina no puede alcanzarse más que por una instrucción GØ SUB. Si se realiza una instrucción GØ TØ en vez de ella, el ordenador no sabrá dónde ir cuando encuentre la instrucción RETURN y se originarán errores.
- Una subrutina debe contener al menos una instrucción RETURN. Ella indica el *fin lógico* de la subrutina y puede ser diferente del *fin físico* como se muestra en el programa siguiente:

```
 900  GØ SUB 2000
      :
2000  LET C = A * B ↑ 1.5
2010  GØ TØ 2040
2020  PRINT X, C
2030  RETURN                    ← fin lógico
2040  LET X = C * K - A
2050  GØ TØ 2020                ← fin físico de la subrutina
      :
```

Una subrutina puede también tener varias instrucciones RETURN correspondiendo a los fines lógicos de las distintas alternativas introducidas por ejemplo con instrucciones IF como en la serie de instrucciones siguiente:

```
 500  GØ SUB 800
      :
 800  IF C > = 10 THEN 830
 810  LET F = C
 820  RETURN
 830  IF C > = 100 THEN 860
 840  LET F = C * 0.95
 850  RETURN
 860  LET F = C * 0.90
 870  RETURN
      :
```

Un programa puede contener varias subrutinas compuestas de un número cualquiera de instrucciones. La posición de estas subrutinas no

tiene ninguna importancia; sin embargo, es preferible agruparlas al final del programa para reducir los riesgos de error.

Una subrutina puede hacer llamada a otra subrutina como en el ejemplo siguiente:

```
100  REM  PRØGRAMA PRINCIPAL
110  GØ SUB 200
120  PRINT A,B,C
130  STØP

200  REM  SUBRUTINA EXTERIOR
210  INPUT A, B
220  GØ SUB 300
230  LET C = C ↑ 2
240  RETURN

300  REM  SUBRUTINA INTERIOR EURE
310  LET C = A + B
320  RETURN

400  END
```

En este caso, el control pasa sucesivamente a las instrucciones 100 - - 110, 200 a 220, 300 a 320, 230 - 240 y 120 - 130.

EJERCICIOS

1. Indicar de las instrucciones siguientes cuáles son las correctas:

```
100  GØ SUB·FACTURACIØN
110  GØ TØ SUB 1000
120  RETURN 500
130  IF X > = 100 THEN GØ SUB 2000
```

2. El programa siguiente debe leer el valor de la variable A, calcular B = 2A e imprimirlo. ¿Qué hace en realidad? ¿Cómo modificarlo?

```
100  INPUT A
110  GØ SUB 200
120  PRINT B

200  LET B = 2 * A
210  RETURN
220  END
```

3. Escribir un programa que contenga las 3 subrutinas siguientes:

- Una que lea 2 variables del teclado.
- Una que calcule la suma A + B.
- Una que imprima el resultado.

El programa debe permitir este cálculo para diferentes pares de datos y pararse cuando A + B sea superior a 10.000.

SOLUCIONES

1. Las cuatro instrucciones son sintácticamente incorrectas:

   - 100 GØ SUB FACTURACION. GØ SUB debe ir seguido del número de la primera línea a ejecutar en la subrutina y no del nombre de esta subrutina.
   - 100 GØ TØ SUB 1000. La instrucción que permite llamar a una subrutina es GØ SUB y no GØ TØ SUB.
   - 120 RETURN 500. El RETURN no debe ir seguido de ninguna expresión.
   - 130 IF X > = 100 THEN GØ SUB 2000. En una instrucción IF-THEN, el THEN debe ir seguido del número de línea a ejecutar, si la condición se cumple y no de GØ SUB ni de GØ TØ.

2. Este programa calcula e imprime B = 2A, pero un número ilimitado de veces. En efecto, se lee A, el control pasa a la línea 200, se calcula B, el control se vuelve a la línea 120 imprimiéndose B.

   Pero ahora el ordenador va a ejecutar la próxima instrucción que se encuentra en la línea 200 y se recalcula B. Al hallar la instrucción RETURN, el control va a volver a la línea cuyo número ha sido puesto en memoria por la última instrucción GØ SUB, es decir, el número 120. B se imprime de nuevo y se vuelve a repetir el ciclo hasta que el usuario detenga manualmente la ejecución de este programa.

   Ciertos sistemas tienen sin embargo la capacidad de detectar este tipo de error. En este caso, cuando se alcanza un RETURN y no ha sido mediante una instrucción GØ SUB, el sistema imprime un mensaje de error y se interrumpe la ejecución.

   Para calcular B e imprimirlo una sola vez, basta introducir una instrucción STØP entre el fin del programa principal y el comienzo de la subrutina. Por ejemplo:

```
130  STØP
```

3.

```
100  REM  PRØGRAMA PRINCIPAL
110  GØ SUB 200
120  GØ SUB 300
130  GØ SUB 400
140  IF C < = 10000 THEN 110
```

```
150  STØP
200  REM  LECTURA
210  INPUT A,B
220  RETURN
300  REM  CALCULØS
310  LET C = A + B
320  RETURN
400  REM  ESCRITURA
410  PRINT C
420  RETURN
500  END
```

CAPITULO III

Realización de un programa en BASIC

En el capítulo II se ha definido el lenguaje BASIC que nos permitía así el escribir programas.

I. LOS COMANDOS DE CONTROL

Deben permitir al usuario indicar al ordenador lo que quiere hacer: escribir un programa, utilizarlo, modificarlo... Los comandos de control no son propios del BASIC y difieren de un constructor a otro. Aquí indicaremos el sistema desarrollado en el Dartmouth College y empleado por General Eléctrica. Los comandos principales de control para los sistemas HP 2000 B, C y F se dan en el apéndice.

En los ejemplos siguientes, los caracteres tecleados por el usuario irán subrayados para diferenciarlos de los escritos por el ordenador. En la hoja del teletipo, los caracteres serán idénticos y por supuesto sin subrayar. Cada línea tecleada por el usuario deberá ir seguida de un retorno de carro.

IDENTIFICACION

Antes de que se pueda escribir o utilizar un programa, el ordenador pide un cierto número de informaciones. Veamos cómo se desarrolla la conversación usuario-ordenador:

HELLØ
USER NUMBER -- L 43217
SYSTEM -- BAS
NEW ØR ØLD -- NEW
NEW FILE NAME -- EJEMPLO
READY

El usuario teclea primero HELLO para indicar al ordenador su deseo de usarlo, el ordenador pide al usuario su número, y el usuario teclea: L 43217. Luego se le debe indicar el lenguaje usado: BAS para BASIC. Pueden estar disponibles otros lenguajes como el FORTRAN o COBOL.

La línea siguiente pregunta si:

- Se quiere emplear un programa ya en memoria, en cuyo caso se escribe ØLD,
- Se quiere escribir un programa nuevo, en este caso se escribe NEW: éste es el caso elegido.

El usuario debe luego identificar el programa nuevo o viejo con un nombre, lo que hace tecleando en la línea siguiente: EJEMPLO.

El ordenador indica que está preparado para trabajar con este programa escribiendo: READY.

ENTRADA DE UN PROGRAMA

El usuario ha expresado su deseo de introducir un nuevo programa tecleando NEW. Después de haber dado un nombre a este programa (EJEMPLO) y haber recibido luz verde del ordenador (READY), no queda más que escribir este programa utilizando el teclado.

Si el usuario quiere borrar todas las instrucciones que ha escrito, basta que teclee SCRATCH. Luego puede volver a escribir el programa: esto puede ser útil cuando se han cometido demasiados errores y es mejor volver a comenzar que corregir todas las líneas.

SALVAR

Si se quiere guardar un programa en memoria auxiliar para utilizarlo más tarde, se puede hacer tecleando SAVE. Para acceder a este programa más tarde, bastará teclear ØLD e indicar el nombre del programa cuando el ordenador lo pida:

```
HELLØ
USER NUMBER -- L 43217
SYSTEM -- BAS
NEW ØR ØLD -- ØLD
ØLD FILE NAME -- EJEMPLO
READY
```

EJECUCION DE UN PROGRAMA

Cuando se ha tecleado el nuevo programa, o enseguida después que el ordenador conteste READY en el caso de un programa viejo, el usuario puede hacer ejecutar su programa tecleando RUN. Los resultados o (mensajes de error) serán impresos por el ordenador. Volviendo a teclear este comando durante la ejecución, se obtendrá la duración de la ejecución hasta este momento.

PARADA DE LA EJECUCION

Puede suceder que se quiera interrumpir la ejecución de un programa. Se puede hacer tecleando STØP. El ordenador estará preparado a hacer otros trabajos, sobre el mismo programa o sobre otro.

BORRADO

Se puede borrar un programa de la memoria, que se puso allí con el comando SAVE, tecleando UNSAVE.

LISTADO DE UN PROGRAMA

El comando LIST permite obtener el listado completo de un programa con una breve cabecera (que da entre otras cosas el nombre del programa). Se puede obtener el mismo listado, pero sin cabecera, con el comando LISTNH.

Si se quiere sólo el listado a partir de la línea 150, se podrá obtener gracias a:

LIST 150 en cuyo caso será precedido de una cabecera

LISTNH 150 en cuyo caso no irá precedido de una cabecera

CAMBIO DE NOMBRE

El nombre del programa que se emplea normalmente, puede cambiarse empleando el comando RENAME. El ordenador pide el nombre del

nuevo programa. Si se emplea el programa EJEMPLO, se le puede dar el nombre ØTRØ así:

```
RENAME
NEW FILE NAME -- AUTRE
READY
```

LONGITUD DE UN PROGRAMA

El programa BASIC debe tener una longitud (medida en caracteres) máxima que depende del ordenador empleado. Si se quiere conocer la longitud del trozo del programa escrito, basta teclear LENGTH.

LISTA DE LOS PROGRAMAS

Para tener la lista de los programas puestos en memoria bajo vuestro número, bastará teclear CATALØG. Si se quiere también la longitud, teclear CASALØG.

Para tener la lista de los programas de la biblioteca es necesario llamar al programa CATLOG *** y hacerle listar así:

```
NEW ØR ØLD -- ØLD
ØLD FILE NAME -- CATLØG ***
READY
LIST
```

PARADA DE LA SESION

Cuando se ha terminado el uso del ordenador, basta teclear BYE o bien GØØD BYE para cortar la comunicación entre el teletipo y el ordenador.

Una instrucción en un programa va siempre precedida por un nombre, el número de la línea. Pero no es así para un comando de control, permitiendo al ordenador conocer la naturaleza de una línea (instrucción o comando) con ver el primer carácter de la línea. Se puede indicar un comando de control con sus tres primeras letras: REN para RENAME, CAT para CATALØG, CAS para CASALØG, etc. Sin embargo, se recomienda escribir los comandos enteros, sobre todo al principio.

II. CORRECCION DE LOS ERRORES

1. ERRORES DE SINTAXIS

Son errores del lenguaje: el usuario ha escrito alguna cosa que no es correcta en BASIC. Estos errores los encuentra el ordenador en la compilación, y la ejecución no puede pues comenzar. Se imprime un mensaje indicando la naturaleza del error y el número de la línea en que se encuentra.

Estos errores son en general fáciles de corregir. El programa imaginario siguiente:

```
100  READ X
110  Y = X ↑ 2
120  LET SQR (Z) = Y
130  GØ TØ 100
140  DATA 4,3?5,6
```

dará estos mensajes de error:

```
NØ ØPERATØR AFTER STATEMENT NUMBER  110
NØ VARIABLE NAME AFTER LET  120
ILLEGAL CHARACTER IN STATEMENT  140
```

El primer mensaje indica que la variable Y en la línea 110 no deberá seguir inmediatamente al número de línea. Esto nos permite ver que se ha omitido la instrucción LET.

El siguiente mensaje indica que lo que sigue a la instrucción LET no es una variable, como debería ser. Se ve que se ha escrito una función a la izquierda del signo "=" lo que no es válido. Si se quiere el valor Z raíz de la ecuación $\sqrt{Z} = Y$, se debe escribir:

```
120  LET  Z = Y ↑ 2
```

El último mensaje indica que hay un carácter ilegal en la línea 140. Se ve rápidamente que se ha tecleado un "?" en lugar de una "," lo que es ciertamente un error en la tecla de mayúsculas/minúsculas.

Para corregir estos errores, basta escribir:

```
110  LET Y = X ↑ 2
120  LET Z = Y ↑ 2
140  DATA 4,3,5,6
```

La nueva línea 110 sustituye automáticamente a la línea con este número en el programa. De la misma manera, si se hubiera querido introducir una instrucción entre las líneas 100 y 110, habría bastado escribir esta instrucción con un número de línea comprendido entre 100 y 110.

La lista de errores de sintaxis dada por el ordenador no es obligatoriamente exhaustiva cuando el número de errores es bastante importante. Por ejemplo se pueden tener 10 mensajes de error. La corrección de estos 10 errores no dará obligatoriamente un programa sintácticamente correcto. Pueden aparecer errores que habían sido enmascarados por los ya recogidos.

2. ERRORES DE LOGICA

Son errores resultantes de la concepción lógica del programa y que no aparecen en la compilación. Algunos, tales como la división por cero, el uso de argumentos ilegales en las funciones, datos insuficientes, etc. son descubiertos por el ordenador y originan mensajes de error. Pero otros errores no son descubiertos por el ordenador y sencillamente dan errores falsos. Para descubrirlos es a menudo necesario probar un programa haciendo los cálculos a mano o con máquina de calcular para diversos grupos de datos y comparando estos resultados con los del ordenador.

Un ejemplo de un error de este tipo es:

```
100  READ A
110  IF A = 0 THEN 140
120  LET S = S + A
130  GØ TØ 100
140  PRINT S
150  READ A
160  IF A < > 0 THEN 120
170  DATA 4,6,1,0
180  DATA 3,2,0,0
190  END
```

Este programa debe hacer la suma de grupos de números. El final de un grupo en la lista de datos se indica con un cero. El final de la lista se indica con dos ceros. Los resultados deberían ser 4 + 6 + 1 = 11 y 3 + 2 = 5. De hecho el ordenador da los valores 11 y 16, lo que deriva de la no inicialización de S. Para el primer grupo de valores, al no estar determinada S toma el valor 0 al principio. Para el segundo grupo, su valor inicial es diferente de cero, e igual a la suma de los números del

grupo precedente, lo que nos conduce a resultados falsos. Esto puede corregirse gracias a la instrucción:

```
155  LET  S = 0
```

En los programas mucho más complicados, puede ser muy difícil descubrir los errores lógicos. Se recurre principalmente a dos métodos:

1. *Simulación a mano.* Para un conjunto sencillo de datos se "simula" al ordenador, es decir, que se sigue el programa instrucción a instrucción, ejecutando estas instrucciones como lo hace el ordenador. Esto permite ver exactamente en qué momento el programa se aleja de lo que se desea obtener.
2. *Trazas.* Son listados del valor de cada variable después de la ejecución de cada instrucción. El principio es el mismo que con la simulación a mano, pero se deja al ordenador el cuidado de calcular los valores de las variables. A menudo basta imprimir sólo los valores de las variables que se juzgan principales en los "puntos estratégicos" del programa, en particular antes de las instrucciones IF. Esto se obtiene ciertamente con instrucciones PRINT que se suprimen una vez corregidos los errores.

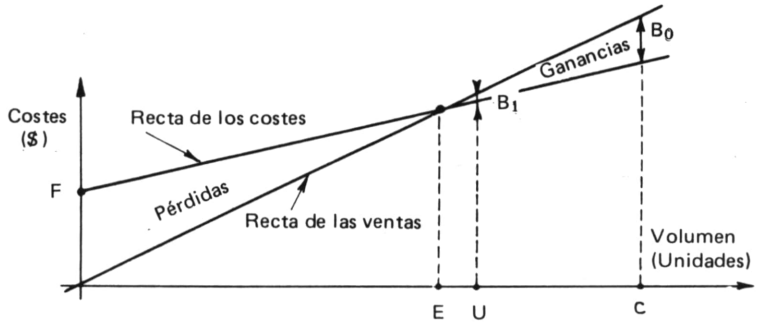
El descubrimiento y corrección de errores ocupa, con frecuencia, una parte considerable del tiempo consumido en la realización del programa. Es más ventajoso a veces consumir más tiempo en la escritura cuidadosa de un programa, que escribirlo rápidamente y esperar que los mensajes de error del ordenador permitieran su acabado.

III. EJEMPLO

Vamos a dar un ejemplo sencillo de la realización de un programa para indicar las diferentes etapas a seguir:

a) El problema

Vamos a estudiar la variación de la rentabilidad de un taller, que produce un solo tipo de piezas, en función del nivel de producción. Supongamos que los costes fijos son F, los costes variables por unidad V y el precio unitario de venta P. Se puede determinar el punto de equilibrio E, los beneficios (o pérdidas) a la capacidad máxima C y al nivel previsto de utilización de la capacidad (U), gracias al diagrama siguiente:



Debemos realizar un programa que calcule E, B1 y B0 a partir de F, V, P, U y C con el fin de obtener rápidamente las variaciones de E, B1 y B0 cuando cambian F y/o V y/o P y/o V y/o C.

b) **Descripción matemática**

Debemos encontrar las relaciones matemáticas que nos permitan obtener E, B1 y B0 a partir de F, V, P, U y C. El estudio del gráfico nos permite escribir enseguida:

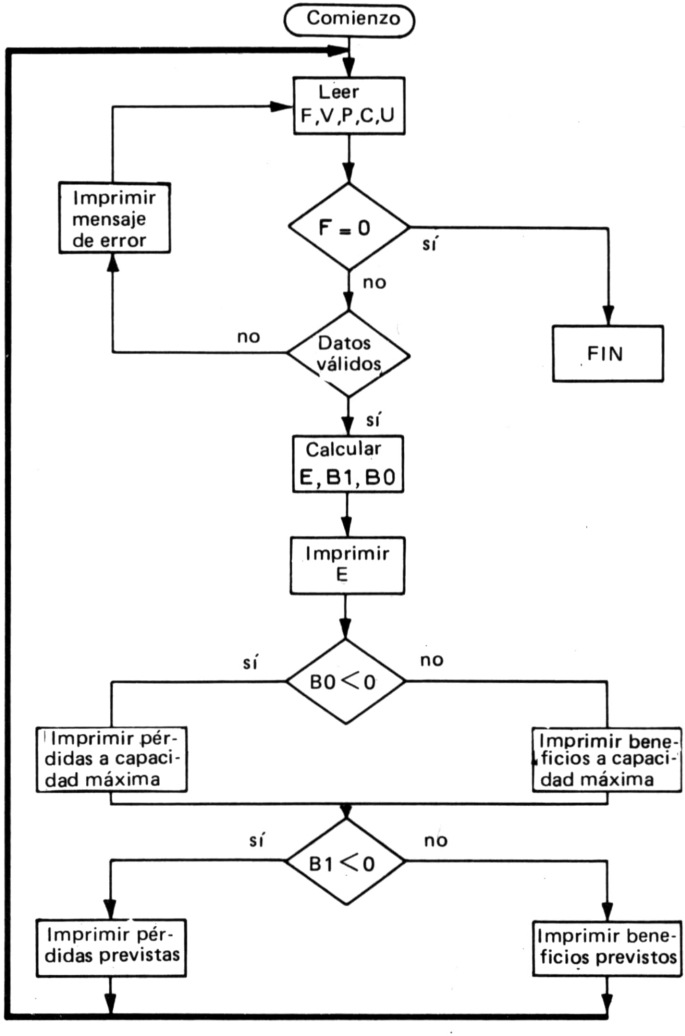
$$E = \frac{F}{P - V}$$
$$B0 = C(P - V) - F$$
$$B1 = U(P - V) - F$$

c) **Organigrama**

El programa debe:

- Leer los datos.
- Verificarlos. En efecto, no se puede tener, por ejemplo una capacidad negativa: si fuera así, el usuario habría hecho un error al entrar el dato y deberá repetirlo.
- Calcular E, B0 y B1
- Imprimir E.
- Imprimir B0 y B1 indicando claramente si se trata de pérdidas o beneficios.
- Saber cuándo se para. Los cálculos se ejecutarán para varios conjuntos de datos. Se detendrá el programa cuando se lea un coste fijo F con un valor de 0.

A partir de esta lista de actividades, se llega después algunos tanteos al organigrama siguiente (1).



(1) Los caracteres subrayados (y por supuesto el programa) han sido tecleados por el usuario.

d) **Programación**

El organigrama anterior permite escribir rápidamente un programa que se teclea en el teletipo o teclado de la forma siguiente:

```
HELLØ
USER NUMBER -- L 43217
SYSTEM -- BAS
NEW ØR ØLD -- NEW
NEW FILE NAME -- PRØCØST
READY
100  PRINT "F,V,P,C,U"
110  INPUT F,V,P,C,U
120  IF F = 0 THEN 320
130  IF F < 0 THEN 300
140  IF V < 0 THEN 300
150  IF C < 0 THEN 300
160  IF U < 0 THEN 300
170  LET E = F/(P - V)
180  LET B0 = C(P - V) - F
190  LET B1 = U(P - V) - F
200  PRINT "PUNTO DE EQUILIBRIO:"; E; "UNIDADES"
210  IF B0 < 0 THEN 260
220  PRINT "BENEFICIO A PLENA CAPACIDAD:"; B0; "$"
230  IF B1 < 0 THEN 280
240  PRINT "BENEFICIO PREVISTO:"; B1; "$"
250  GØ TØ 100
260  PRINT "PERDIDAS A PLENA CAPACIDAD"; - B0; "$"
270  GØ TØ 230
280  PRINT "PERDIDAS PREVISTAS"; - B1; "$"
290  GØ TØ 100
300  PRINT "ERROR EN LOS DATOS. REPETIR"
310  GØ TØ 100
320  END
```

e) **Corrección de errores**

Para saber si existen errores en este programa, basta teclear RUN después de la última instrucción. Veamos la respuesta del ordenador:

```
RUN
WAIT

PRØCØST        15 : 31        PARIS2   25/05/70

ILLEGAL VARIABLE NAME 180
ILLEGAL VARIABLE NAME 190
ILLEGAL STATEMENT 250

USED : 0 SEC.
```

Los errores en las líneas 180 y 190 se derivan del hecho de que se han escrito las fórmulas matemáticas olvidando que la multiplicación no es implícita en BASIC, se debe añadir el símbolo "*". El error de la línea 250 es más difícil de ver y pide una buena vista: La segunda letra del "GØ" no es una Ø sino un 0. Este error de tecleo es muy corriente, lo mismo que el empleo de "I" en vez de "1"

Se pueden corregir estos errores y tratar de ejecutar el programa como sigue:

```
180  LET B0 = C*(P - V) - F
190  LET B1 = U*(P - V) - F
250  GØ TØ 100
RUN
WAIT


PRØCØST        15: 32        PARIS 2  25/05/70

F,V,P,C,U
?40000,2,6,20000,12000
PUNTO DE EQUILIBRIO:  10000          UNIDADES
BENEFICIO A PLENA CAPACIDAD:          4000          $
BENEFICIO PREVISTO:  8000          $
F, V, P, C, U
?50000, 1, 11, 20000, 15000
PUNTO DE EQUILIBRIO:  5000          UNIDADES
BENEFICIO A PLENA CAPACIDAD:          150000          $
BENEFICIO PREVISTO:  100000          $
F, V, P, C, U
?0, 0, 0, 0, 0


USED          2.05  SEC.
```

f) **Mejoras**

Se puede pensar en mejorar la presentación:

- Indicando al usuario del programa lo que significan F, V, P, C, U;
- Separando con una línea en blanco los resultados que se refieren a otros datos.

Esto se puede obtener tecleando las nuevas instrucciones:

```
 40  PRINT  "F  =  CØSTES FIJOS (EN $)"
 50  PRINT  "V  =  COSTES VARIABLES EN $ / UNIDAD"
 60  PRINT  "P  =  PRECIØ (EN $ / UNIDAD)"
 70  PRINT  "C  =  CAPACIDAD MAXIMA  (EN UNIDADES)"
 80  PRINT  "U  =  CAPACIDAD EMPLEADA  (EN UNIDADES)"
 90  PRINT
100  PRINT
105  PRINT  "F, V, P, C, U"
RUN
WAIT.


PRØCØST        15: 35        PARIS 2   25/05/70

F  =  CØSTES FIJOS  (EN $)
V  =  CØSTES VARIABLES EN  $ / UNIDAD
P  =  PRECIØ (EN  $ / UNIDAD)
C  =  CAPACIDAD MAXIMA (EN UNIDADES)
U  =  CAPACIDAD UTILIZADA (EN UNIDADES)

F, V, P, C, U
?40000, 2, 6, 20000, 12000
PUNTØ DE EQUILIBRIO:   10000          UNIDADES
BENEFICIO A PLENA CAPACIDAD:          40000          $
BENEFICIO PREVISTO:       8000    $

F, V, P, C, U
?100000, 5, 10, 15000, 10000
PUNTØ DE EQUILIBRIO:      20000          UNIDADES
PERDIDAS A PLENA CAPACIDAD:           25000          $
PERDIDAS PREVISTAS        50000    $

F, V, P, C, U
?0, 0, 0, 0, 0


USED          2:20     SEC
```

Después de haber verificado la exactitud de los cálculos (a mano o con calculadora) para estos datos, el usuario puede:

– Pedir un listado completo de las instrucciones,

– Guardar el programa en memoria auxiliar (para usarlo más tarde).

– Cortar la comunicación con el ordenador, como sigue:

```
LIST

PRØCØST        15.40        PARIS 2  25/05/80

 40  PRINT "F = CØSTES FIJOS (EN $)"
 50  PRINT "V = CØSTES VARIABLES EN $/ UNIDAD"
 60  PRINT "P = PRECIØ (EN $/ UNIDAD)"
 70  PRINT "C = CAPACIDAD MAXIMA (EN UNIDADES)"
 80  PRINT "U = CAPACIDAD UTILIZADA (EN UNIDADES)"
 90  PRINT
100  PRINT
105  PRINT "F,V,C,P,U"
110  INPUT F,V,C,P,U
120  IF F = 0 THEN 320
130  IF F < 0 THEN 300
140  IF V < 0 THEN 300
150  IF C < 0 THEN 300
160  IF U < 0 THEN 300
170  LET E = F/(P - V)
180  LET B0 = C*(P - V) - F
190  LET B1 = U*(P - V) - F
200  PRINT "PUNTØ DE EQUILIBRIO:"; E; "UNIDADES"
210  IF B0 < 0 THEN 260
220  PRINT "BENEFICIO A PLENA CAPACIDAD:"; B0; "$"
230  IF B1 < 0 THEN 280
240  PRINT "BENEFICIO PREVISTO:"; B1; "$"
250  GØ TØ 100
260  PRINT "PERDIDAS A PLENA CAPACIDAD:"; -B0; "$"
270  GØ TØ 230
280  PRINT PERDIDAS PREVISTAS"; - B1; "$"
290  GØ TØ 100
300  PRINT "ERROR EN LOS DATOS. REPETIR"
310  GØ TØ 100
320  END

SAVE
READY
BYE
*** ØFF AT 15.42 MØN 25/05/80
```

CAPITULO IV

Programas útiles

Esta parte contiene un cierto número de programas útiles en diferentes dominios: estadística, investigación operativa, finanzas y producción. El objetivo de este capítulo es doble:

1. Suministrar un cierto número de ejemplos de programas escritos en BASIC. El lector puede, como ejercicio, tratar de reconstruir estos programas a partir de la hoja de resultados y de la especificación de los datos.
2. Permitir al lector emplear el ordenador para aplicaciones muy normales, sin tener que volver a escribir él mismo programas clásicos.

Cada programa se presentará de la forma siguiente:

1. El problema a resolver. Cuando sean necesarios conocimientos particulares para su resolución, se harán referencias a obras especializadas.
2. Esbozo del método usado.
3. Presentación de los datos. Los programas se han hecho conversacionales cuando la lista de datos es corta. Cuando esta lista puede ser grande, ha parecido preferible incluirla en el programa.
4. Listado del programa y hoja de resultados para un conjunto de datos.

Los comentarios incluídos en los programas permiten una comprensión cómoda y facilitan las modificaciones.

I. CLASIFICACION DE DATOS ESTADISTICOS

a) **Problema** (*)

Este programa debe transformar una serie numérica no ordenada en una serie ordenada. Deberá imprimirse una tabla de frecuencias indicando para cada clase el límite inferior, el límite superior, la frecuencia absoluta y la frecuencia relativa acumulada.

b) **Método**

Las frecuencias absolutas se calculan durante la lectura de datos. Las frecuencias relativas y relativas acumuladas se calculan cuando se imprimen los resultados.

c) **Presentación de los datos**

Deben escribirse en el programa. El usuario debe indicar para cada serie de datos:

- El número de clases,
- El límite superior de la primera clase,
- El intervalo de las clases,
- Los datos de la serie numérica,
- El número 99999 para indicar el fin de la serie.

Varias series de datos pueden escribirse consecutivamente. El número 99999 debe ponerse para indicar el fin de la lista de datos.

El número de valores en una serie está limitado a 100.

d) **Programa y hojas de resultados** (Pág. 103, 104)

II. ANALISIS DE DATOS ESTADISTICOS (*)

a) **Problema**

Este problema debe indicar el número de datos, el valor máximo, el valor mínimo, la mediana, la media, la varianza y la desviación típica de una serie numérica.

(*) Ver la obra: Calot "Curso de estadística descriptiva". 3ª ed. Editorial Paraninfo, 1982, Madrid (España).

b) **Método**

Al leer los datos se calcula la suma de los datos, la suma de los cuadrados y los valores máximo y mínimo. La media $\bar{X}$ y la varianza V se calculan con las fórmulas:

$$\bar{X} = \frac{\Sigma Xi}{N}$$

$$V = \frac{N\Sigma Xi^2 - (\Sigma Xi)^2}{N(N-1)}$$

siendo:

N	El número de los datos,
ΣX_i	Es la suma de los datos,
ΣX_i^2	Es la suma de los cuadrados de los datos.

La desviación típica es la raiz cuadrada de la varianza.

c) **Presentación de los datos**

Deben imprimirse en el programa. Basta indicar consecutivamente cada serie numérica acabada con 99999. Dos series numéricas deben ir separadas con un 0. El último 99999 de la lista de datos debe ir seguido de un número cualquiera distinto de 0.

El número de valores en una serie está limitado a 100.

d) **Programa y hoja de resultados** (Pág. 105)

III. REGRESION LINEAL (*)

a) **Problema**

Dado un conjunto de puntos definido con N pares de valores X*i* e Y*i*, la regresión lineal permite determinar la recta Y = AX + B que se ajuste lo mejor posible a este conjunto. El coeficiente de correlación R es una medida del ajuste de la recta a los datos. La desviación típica de la esti-

(*) Idem a la nota de la pág. 100.

mación de E es una medida de la dispersión de valores Y alrededor de la recta.

Los valores A, B, R y E deben calcularse.

b) **Método**

Se usan las fórmulas siguientes:

$$A = \frac{N\Sigma Xi Yi - (\Sigma Xi)(\Sigma Yi)}{N\Sigma Xi^2 - (\Sigma Xi)^2}$$

$$B = \bar{Y} - A\bar{X}$$

$$R = \frac{N\Sigma Xi Yi - (\Sigma Xi)(\Sigma Yi)}{\sqrt{N\Sigma Xi^2 - (\Sigma Xi)^2}.\sqrt{N\Sigma Yi^2 - (\Sigma Yi)^2}}$$

$$E = \sqrt{\frac{\Sigma Yi^2 - B\Sigma Yi - A\Sigma Xi Yi}{N - 2}}$$

Los valores ΣXi, ΣYi, ΣY_i^2, ΣX_i^2 y $\Sigma Xi Yi$ se calculan durante la lectura.

c) **Presentación de los datos**

Deben ser impresos por el programa. Los pares X, Y formando el conjunto de puntos se escribe consecutivamente. Cada serie de datos debe acabarse con 99999. Dos series deben ir separadas con un 0. El último 99999 de la lista de datos debe ir seguido de un número cualquiera diferente de 0.

El número de pares de valores se limita a 100.

d) **Programa y hoja de resultados** (Pág. 107).

IV. CALCULO DE UNA AMORTIZACION DECRECIENTE

a) **Problema**

Sea una inversión I a amortizar en N años. Esto se puede hacer linealmente a razón de 100%/N por año. Otro método llamado de amor-

PROGRAMA

```
100  PRINT "CLASIFICACION DE DATOS ESTADISTICOS"
110  PRINT
120REM: LECTURA DEL NUMERO DE CLASES C
130  READ C
140REM: SI C = -99999 FIN DE LA LISTA DE DATOS
150  IF C = -99999 THEN 630
160REM: LECTURA DE P Y L
170  READ P, L
180  DIM N (20)
190REM: LECTURA DE DATOS Y CALCULO DE LAS FRECUENCIAS ABSOLUTAS
200  FØR I = 1 TØ 100
210     READ D
220     IF D = 99999 THEN 320
230     IF D > = P THEN 260
240     LET N (1) = N (1) + 1
250     GØ TØ 310
260     IF D < P + (C -2) * L THEN 290
270     LET N (C) = N (C) + 1
280     GØ TØ 310
290     LET K = INT ((D - P) / L) + 2
300  LET N (K) = N (K) + 1
310  NEXT I
320  LET I = I - 1
330REM: ESCRITURA DE CABECERAS
340  PRINT
350  PRINT
360  PRINT "NUMERO DE DATOS:"; I
370  PRINT
380  PRINT "  ", "  ", "TABLA DE FRECUENCIAS"
390  PRINT
400  PRINT "  LIMITE","    LIMITE", "FRECUENCIAS", "FRECUENCIAS", "FRECUENCIAS"
410  PRINT "INFERIOR", "SUPERIOR", "ABSOLUTAS", "RELATIVAS", "RELATIVAS"
420  PRINT "  ", "  ", "  ", "  ", "  ", "ACUMULADAS"
430REM: CLASE 1
440  LET B = N (1)/I
450  PRINT "-INFINITO", P, N (1), B, B,
460  LET S = B
470  LET N (1) = 0
480REM: "CLASES INTERMEDIAS"
490  FØR J = 2 TØ C - 1
500     LET B = N (J) /I
510     LET S = S + B
520     PRINT P + L * (J - 2), P + L * (J - 1), N (J), B, S
530  LET N (J) = 0
540  NEXT J
550REM: CLASE C
560  LET B = N (C)/I
570  PRINT P + L * (C - 2), "+ INFINITO", N (C), B, S + B
580  LET N (C) = 00
590  GØ TØ 110
600  DATA 6,2,2,2,6,4,5,7,8,3,5,4,3,6, 5,6,4,9,99999
610  DATA 5, 10, 5, 1, 18,35,50,22,17,9,8,12,15,99999
620  DATA -99999
630  END
```

HOJA DE RESULTADOS

CLASIFICACIØN DE DATØS ESTADISTICØS

NUMERØ DE DATØS: 15

TABLA DE FRECUENCIAS

LIMITE INFERIOR	LIMITE SUPERIOR	FRECUENCIAS ABSOLUTAS	FRECUENCIAS RELATIVAS	FRECUENCIAS RELATIVAS ACUMULADAS
- INFINITO	2	0	0	0
2	4	3	.2	.2
4	6	6	.4	.6
6	8	4	.266667	.866667
8	10	2	.133333	1.
10	+ INFINITO	0	0	1.

NUMERØ DE DATØS: 10

TABLA DE FRECUENCIAS

LIMITE INFERIØR	LIMITE SUPERIØR	FRECUENCIAS ABSØLUTAS	FRECUENCIAS RELATIVAS	FRECUENCIAS RELATIVAS ACUMULADAS
- INFINITO	10	3	.3	.3
10	15	1	.1	.4
15	20	3	.3	.7
20	25	1	.1	.8
25	+ INFINITO	2	.2	1.

PROGRAMA

```
100 PRINT "ANALISIS DE DATOS ESTADISTICOS"
110 PRINT
120REM: INICIALIZACION
130 LET S1 = 0
140 LET S2 = 0
150 LET D1 = 999999
160 LET D2 = -999999
170REM: "LECTURA Y CALCULO DE S1 y S2
180 FØR N = 1 TØ 100
190   READ D
200   IF D = 99999 THEN 280
210   LET S1 = S1 + D
220   LET S2 = S2 + D * D
230   IF D > = D1 THEN 250
240   LET D1 = D
250   IF D <= D2 THEN 270
260   LET D2 = D
270 NEXT N
280 LET N = N - 1
290REM: CALCULØ
300 LET M = (D1 + D2) /2
310 LET Y = S1/N
320 LET V = (N * S2 - S1 * S1)/ N /(N - 1)
330 LET E = SQR (V)
340REM: ESCRITURA
350 PRINT
360 PRINT "NUMERØ DE DATØS"; N
370 PRINT "VALØR MAXIMØ:"; D2
380 PRINT "VALØR MINIMØ:"; D1
390 PRINT "MEDIANA:"; M
400 PRINT "MEDIA:"; Y
410 PRINT "VARIANZA:"; V
420 PRINT "DESVIACIØN TIPICA:"; E
430REM: PRUEBA FIN DE LA LISTA DE DATØS
440 READ R
450 IF R = 0 THEN 110
460 DATA 7, 4, 8, 9, 3, 1, 6, 99999, 0
470 DATA 2, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 99999, 1
480 END
```

HOJA DE RESULTADOS

```
ANALISIS DE DATOS ESTADISTICOS


NUMERO DE DATOS: 7
VALOR MAXIMO: 9
VALOR MINIMO: 1
MEDIANA: 5
MEDIA; 5.42857
VARIANZA: 8.28571
DESVIACION TIPICA: 2.87849

NUMERO DE DATOS: 15
VALOR MAXIMO: 8
VALOR MINIMO: 2
MEDIANA: 5
MEDIA; 6.06667
VARIANZA: 2.20952
DESVIACION TIPICA: 1.48645
```

tización decreciente, consiste en amortizar A % del valor contable cada año, hasta el año L-1, y luego amortizar el resto linealmente entre los años L y N. La fracción A se escoge igual a 200 % N.

Dados los tres valores I, L y N se pide calcular la amortización anual y el valor contable para los N años.

b) **Método**

Se calcula primero la amortización anual y el valor contable y se imprimen, para el período en que se aplica la razón decreciente (entre los años 1 y L-1). El valor contable en el año L-1 permite calcular la amortización lineal entre los años L y N.

c) **Presentación de los datos**

El usuario debe dar los valores de I, N y L al ejecutarse el programa. Después de la impresión de los resultados, se pide si se desea modificar alguno de estos valores o bien parar la ejecución.

d) **Programa y hoja de resultados** (Pág. 108, 109)

V. CALCULO DEL VALOR ACTUALIZADO

a) **Problema**

Se ha previsto una inversión con una serie de ingresos anuales correspondientes. Se trata de calcular el importe que en el momento de la inversión equivale a esta serie de ingresos, para lo que se supone conocida una tasa de actualización (coste de capital, tasa de rentabilidad media de las inversiones...)

b) **Método**

El valor actualizado se obtiene con la fórmula:

$$V = \sum_{j=0}^{N} \frac{Xj}{(1 + I)^j}$$

siendo Xj el ingreso en el año j,
I la tasa de actualización,
N la duración del proyecto.

PROGRAMA

```
100 PRINT "REGRESIØN LINEAL"
110 PRINT
120REM: INICIALIZACIØN
130 LET X1 = 0
140 LET Y1 = 0
150 LET X2 = 0
160 LET Y2 = 0
170 LET P = 0
180REM: LECTURA DE DATØS Y CALCULØ DE X1, Y1, X2, Y2 Y P
190 FØR N = 1 TØ 100
200   READ X
210   IF X = 99999 THEN 290
220   READ Y
230   LET X1 = X1 + X
240   LET Y1 = Y1 + X
250   LET X2 = X2 + X * X
260   LET Y2 = Y2 + Y * Y
270   LET P = P + X * Y
280 NEXT N
290 LET N = N - 1
300REM: CALCULOS
310 LET M1 = X1/N
320 LET M2 = Y1/N
330 LET A = (N * P - X1 * Y1)/(N * X2 - X1 * X1)
340 LET B = M2 - A * M1
350 LET R = ((N * P) - X1 * Y1) / SQR (N * N2 - X1 * X1) / SQR (N * Y2 - Y1 * Y1)
360 LET E = SQR ((Y2 - B * Y1 - A * P) / (N - 2) / (N - 2))
370REM: ESCRITURA
380 PRINT
390 PRINT "N ="; N
400 PRINT "A ="; A
410 PRINT "B ="; B
420 PRINT "R ="; R
430 PRINT "E ="; E
440REM: PRUEBA FIN DE LA LISTA DE DATØS
450 READ X
460 IF X = 0 THEN 110
470 DATA 2, 10, 3, 16, 5, 20, 6, 30, 99999, 0
480 DATA 10, 10, 20, 10, 60, 30, 100, 40, 210, 110, 99999, 1
490 END
```

HOJA DE RESULTADOS

```
REGRESION LINEAL

N = 4
A = 4.4
B = 1.4
R = .955619
E = 2.14476

N = 5
A = .503817
B = .305343
R = .988937
E = 4.07729
```

PROGRAMA

```
100  PRINT "AMØRTIZACIØN  DECRECIENTE"
110REM: ENTRADA DE LØS VALØRES DE I, N Y L
120  PRINT
130  PRINT "INDICAR LØS VALØRES DE I, N  Y L"
140  INPUT I, N, L
150REM; ESCRITURA DE CABECERAS
160  PRINT
170  PRINT "ANUAL", "AMØRTIZACIØN", "VALØR CØNTABLE"
180  LET V = I
190REM: PERIØDØ DE AMØRTIZACIØN DECRECIENTE
200  FØR  J = 1  TØ  L - 1
210     LET  A = V * 2/N
220     LET  V = V - A
230     PRINT  J, A, V
240  NEXT J
250REM: PERIØDØ DE AMØRTIZACIØN  LINEAL
260  LET  A = V / (N - L + 1)
270  FØR  J = L  TØ  N
280     LET  V = V - A
290     PRINT  J, A, V
300  NEXT J
310REM: MØDIFICACIØN DE LØS DATØS?
320  PRINT
330  PRINT "QUIERE  VD. MODIFICAR I, NØ L?  1 = SI,  2 = NO"
340  INPUT  R
350  IF  R = 1  THEN  130
360  IF  R = 2  THEN  390
370  PRINT "VALIDØ SØLØ 1Ø 2. REPETIR"
380  GØ TØ 340
390  END
```

HOJA DE RESULTADOS

AMØRTIZACIØN DECRECIENTE

INDICAR LØS VALØRES DE I, N, L
? 10000, 10, 1

ANUAL	AMØRTIZACIØN	VALØR CØNTABLE
1	1000	9000
2	1000	8000
3	1000	7000
4	1000	6000
5	1000	5000
6	1000	4000
7	1000	3000
8	1000	2000
9	1000	1000
10	1000	0

QUIERE VD. MØDIFICAR I, N Ø L? 1 = SI, 2 = NØ
? 1
INDICAR LOS VALORES DE I, N y L
? 100000, 10, 5

ANUAL	AMØRTIZACIØN	VALØR CØNTABLE
1	20000	80000
2	16000	64000
3	12800	51200
4	10240	40960
5	6826.67	34133.3
6	6826.67	27306.7
7	6826.67	20480.
8	6826.67	13653.3
9	6826.67	6826.67
10	6826.67	-6.10352E-05

QUIERE VD. MØDIFICAR I, N Ø L? 1 = SI, 2 = NØ
? 1
INDICAR LOS VALORES DE I, N y L
? 10000, 10, 10

ANUAL	AMØRTIZACIØN	VALØR CØNTABLE
1	2000	8000
2	1600	6400
3	1280	5120
4	1024	4096
5	819.2	3276.8
6	655.36	2621.44
7	524.288	2097.15
8	419.43	1677.72
9	335.544	1342.18
10	1342.18	0

QUIERE VD. MØDIFICAR I, N Ø L? 1 = SI, 2 = NØ
? 2

Esta fórmula supone que la inversión se hace al final del año 0, y que los ingresos se hacen globalmente al final de cada año y que se actualizan al final de cada año (o al comienzo del año 1).

c) **Presentación de los datos**

Estan contenidos en el programa. Para un proyecto de inversión, se deben dar los datos siguientes:

- Los ingresos anuales considerados,
- El número 99999 indicando el final de la lista de ingresos,
- Las diferentes tasas de actualización para las que se quiere conocer el valor actualizado,
- Un número negativo indicando el fin de la lista de tasas de actualización.

Dos series de datos correspondientes a dos proyectos de inversión diferentes van separadas por un cero.

La lista de datos debe acabarse con un número diferente de cero. La duración de un proyecto se limita a 50 años.

d) **Programa y hoja de resultados** (pág. 112)

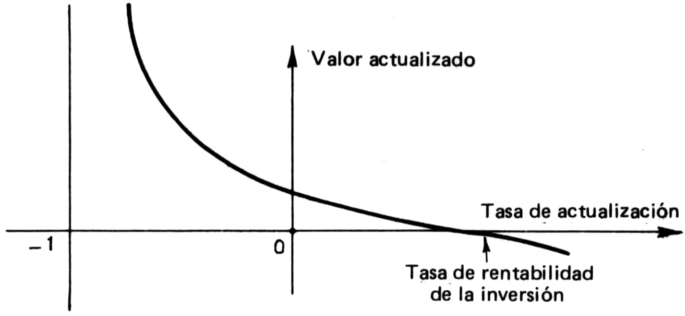
VI. CALCULO DE LA TASA DE RENTABILIDAD DE UNA INVERSION

a) **Problema**

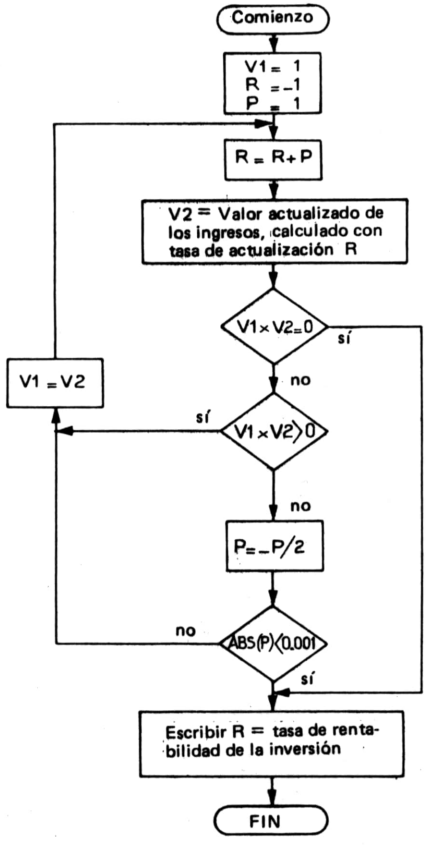
Se trata de calcular la tasa de rentabilidad de una inversión, es decir, la tasa de actualización tal que el valor actualizado de los ingresos correspondientes a este proyecto de inversión sea nulo.

b) **Método**

La variación del valor actualizado de los ingresos en función de la tasa de actualización se puede representar como sigue:



La tasa de rentabilidad de la inversión puede ser calculada gracias a un procedimiento representado por el organigrama adjunto.



PROGRAMA

```
100  PRINT "VALØR ACTUALIZADØ"
110  PRINT
120  DIM X (50)
130REM: LECTURA DE LØS INGRESØS
140  FØR J = 1 TØ 50
150     READ X (J)
160     IF X (J) = 99999 THEN 190
170  NEXT J
180REM: DURACIØN DEL PRØYECTØ
190  LET N = J - 1
200  PRINT "DURACIØN DEL PRØYECTØ:"; N - 1; "AÑOS"
210REM: INICIALIZACIØN
220  LET V = 0
230REM: LECTURA TASA ACTUALIZACIØN
240REM: FIN DE LA LISTA DE TASAS DE ACTUALIZACIØN?
250  READ I
260  IF I < 0 THEN 360
270REØ: CALCULØ DEL VALØR ACTUALIZADØ
280  FØR J = 1 TØ N
290     LET V = V + X (J) / (1 + I) ↑ (J - 1)
300  NEXT J
310REM: ESCRITURA
320  PRINT "TASA DE ACTUALIZACIØN"; I
330  PRINT "VALØR ACTUALIZADØ", V
340  GØ TØ 220
350REM: FIN DE LA EJECUCIØN
360  READ R
370  IF R = 0 THEN 110
380  DATA -250, 100, 100, 100, 100, 9999, 0, 0.10, 0.15, 0.20
390  DATA - 0.10, 0
400  DATA -5000, 2560, 3000, 1000, 99999, 0, 0.20, 0.50
410  DATA -0.10, 1
420  END
```

HOJA DE RESULTADOS

```
VALØR ACTUALIZADØ

DURACIØN DEL PRØYECTØ : 4 AÑOS
TASA DE ACTUALIZACIØN: 0
VALØR ACTUALIZADØ: 150
TASA DE ACTUALIZACIØN: .1
VALØR ACTUALIZADØ: 66.9865
TASA DE ACTUALIZACIØN: .15
VALØR ACTUALIZADØ: 35.4978
TASA DE ACTUALIZACIØN: .2
VALØR ACTUALIZADØ: 8.87346

DURACIØN DEL PRØYECTØ: 3 AÑOS
TASA DE ACTUALIZACIØN: 0
VALØR ACTUALIZADØ: 1560
TASA DE ACTUALIZACIØN: .2
VALØR ACTUALIZADØ: -20T.63
TASA DE ACTUALIZACIØN: .5
VALØR ACTUALIZADØ: -1663.7
```

CALCULO DE LA TASA DE RENTABILIDAD DE UNA INVERSION

PROGRAMA

```
100  PRINT "TASA DE RENTABILIDAD DE UNA INVERSIØN
110  PRINT
120  DIM X (50)
130REM: LECTURA DE INGRESØS
140  FØR J = 1 TØ 50
150     READ X (J)
160     IF X (J) = 99999 THEN 190
170  NEXT J
180REM: INICIALIZACIØN
190  LET N = J - 1
200  LET V1 = 1
210  LET R = 1
220  LET P = 1
230REM: CALCULØ DE T . R . I .
240  LET V2 = 0
250  LET R = R + P
260REM: CALCULØ DEL VALØR ACTUALIZADØ
270  FØR J = 1 TØ N
280     LET V2 = V2 + X (J) / (1 + R) ↑ (J - 1)
290  NEXT J
300REM: VALØR ACTUALIZADØ = 0?
310  IF V1 * V2 = 0 THEN 390
320REM: VALØR ACTUALIZADØ CAMBIA DE SIGNØ?
330  IF V1 * V2 > 0 THEN 360
340  LET P = - P/2
350  IF ABS (P) < 0.001 THEN 390
360  LET V1 = V2
370  GØ TØ 240
380REM: ESCRITURA
390  PRINT "DURACIØN DEL PRØYECTØ"; N - 1
400  PRINT "TASA DE RENTABILIDAD"; R
410REM: FIN DE LA LISTA DE DATØS
420  READ R
430  IF R = 0 THEN 110
440  DATA - 1000, 500, 600, 800, 900, 99999, 0
450  DATA - 5000, 500, 600, 800, 900, 99999, 1
460  END
```

HOJA DE RESULTADOS

```
TASA DE RENTABILIDAD DE UNA INVERSIØN

DURACIØN DEL PRØYECTØ: 4
TASA DE RENTABILIDAD: .509766

DURACIØN DEL PRØYECTØ: 4
TASA DE RENTABILIDAD: .183594
```

c) Presentación de los datos

Deben estar contenidos en el programa. Basta escribir la lista de ingresos anuales correspondientes a un proyecto de inversión seguidos de 99999. Dos series de datos deben estar separados por un cero. El último 99999 de la lista debe ir seguido por un número cualquiera diferente de cero.

d) **Programa y hoja de resultados** (Pág. 113).

PROGRAMA

```
100 PRINT "CANTIDAD ECØNØMICA DE PEDIDØ"
110 PRINT
120REM: ENTRADA DE D, S, V, K1 Y K2
130 PRINT "INDICAR D, S, V, K1 Y K2"
140 INPUT D, S, V, K1, K2
150REM: CALCULØ Y ESCRITURA DE VALØRES ØPTIMØS
160 LET QO = INT (SQR (2 * D * S/ (K1 + K2 * V)))
170 LET CO = INT ((QO * (K1 + K2 * V) / 2 + D * S / QO) * 100) / 100
180 PRINT "CANTIDAD ECØNØMICA DE PEDIDØ"; QO
190 PRINT "CØSTE TOTAL MINIMØ:", CO
200REM: CALCULØ Y ESCRITURA DE VARIACIØNES ALREDEDØR DEL ØPTIMØ
210 PRINT
220 PRINT "VARIACIØNES ALREDEDØR DEL ØPTIMØ"
230 PRINT "Q", "C" CØM", "C STØCK", "C TØTAL", PØRC. VARIAC."
240 FØR J = - 0,8 TØ 1 STEP 0.2
250    LET Q = QO + INT (QO * J)
260    LET C1 = INT (D * S/Q * 100) / 100
270    LET C2 = INT (Q * (K1 + K2 * V) /2 * 100) / 100
280    LET P = INT ((C1 + C2 - CO) / CO * 100)
290    PRINT Q, C1, C2, C1 + C2, P
300 NEXT J
310REM: MØDIFICACIØN DE LØS PARAMETRØS
320 PRINT
330 PRINT "QUIERE VD. MØDIFICAR LØS PARAMETRØS"?
340 PRINT "1 = SI, 2 = NO"
350 INPUT R
360 IF R = 1 THEN 110
370 IF R = 2 THEN 400
380 PRINT "VALIDØ SØLØ 1 Ø 2. REPETIR"
390 GØ TØ 350
400 END
```

VII. CALCULO DE LOS LOTES ECONOMICOS DE PEDIDO

a) **Problema**

Este programa debe calcular cuál es la cantidad de pedido óptima para un cierto producto guardado en almacén, con el fin de minimizar los costes anuales totales: costes de pedido, más costes de almacenamiento. Los costes de pedido y de almacenamiento, el coste total y la variación en porcentaje del coste total con relación a su mínimo deben calcularse para cantidades de pedido cercanas a la cantidad económica.

b) **Método**

La cantidad económica de pedido Q_0 y el coste mínimo correspondiente C_0 se obtienen con la ayuda de las fórmulas:

$$Q_0 = \sqrt{\frac{2 \times D \times S}{K1 + K2 \times V}}$$

$$C_0 = \sqrt{2 \times D \times S \times (K1 + K2 \times V)}$$

siendo: D = cantidad anual consumida,
S = coste de preparación de un pedido,
K_1 = componente del coste de almacenamiento unitario proporcional a la cantidad,
K_2 = componente del coste de almacenamiento unitario proporcional al valor (expresado en % del valor),
V = valor unitario de la mercancía almacenada.

Los diferentes costes se calculan para valores de la cantidad de pedido variando entre el 20 % y el 200 % de la cantidad económica como un paso del 20 %

Los costes anuales de pedido y de almacenamiento se obtienen gracias a las fórmulas:

$$\text{coste de pedido} = \frac{D \times S}{Q}$$

$$\text{coste de almacenamiento} = \frac{Q}{2} (K1 + K2 \times V)$$

siendo Q la cantidad de pedido.

HOJA DE RESULTADOS

```
CANTIDAD ECØNØMICA DE PEDIDØ

INDICAR D, S, V, K1 Y K2
? 60000, 12, 0.8, 0.04, 0.15
CANTIDAD ECØNØMICA DE PEDIDØ: 3000
CØSTE TØTAL MINIMØ:              479.99

VARIACIØNES ALREDEDØR DEL ØPTIMØ

   Q          C CØM       C STØCK      C TØTAL      PORC VARIAC
 600          1200         47.99       1247.99          160
1199          600.5        95.91        696.41           45
1799          400.22      143.91        544.13           13
2399          300.12      191.91        492.03            2
2999          240.08      239.91        479.99            0
3599          200.05      287.91        487.96            1
4199          171.46      335.91        507.37            5
4799          150.03      383.11        533.94           11
5399          133.35      431.91        565.26           17
5999          120.02      479.91        599.93           24

QUIERE MØDIFICAR LØS PARAMETRØS?
1 = SI, 2 = NO
? 1

INDICAR D, S, V, K1 y K2
? 80000, 12, 0.8, 0.04, 0.15
CANTIDAD ECØNØMICA DE PEDIDØ: 3464
CØSTE TØTAL MINIMØ:              554.25

VARIACIØNES ALREDEDØR DEL ØPTIMØ

   Q          C CØM       C STØCK      C TØTAL      PORC VARIAC
 692          1387.28      55.35       1442.63          160
1385          693.14      110.79        803.93           45
2078          461.98      166.23        628.21           13
2771          346.44      221.67        568.11            2
3463          277,21      277.03        554.24           -1
4156          230.99      332.47        563.46            1
4849          197.97      387.91        585.88            5
5542          173.22      443.35        616.57           11
6235          153.96      498.79        652.75           17
6927          138.58      554.15        692.73           24

QUIERE VD. MØDIFICAR LØS PARAMETRØS?
1 = SI, 2 = NO
? 2
```

c) **Presentación de los datos**

Este programa es conversacional. El usuario debe indicar en la ejecución los valores de D, S, V, K1, K2. Después de la escritura de resultados, se pide si se quiere continuar, cambiando algún valor.

d) **Programa y hoja de resultados** (Pág. 114, 116)

VIII. CAMINO CRITICO

a) **Problema**

Dada una red PERT cuyos nudos van numerados secuencialmente, este programa calcula:

- El tiempo más pronto o más tarde para el comienzo y fin de cada actividad.
- Los márgenes totales, libres e independientes para cada actividad.

b) **Método**

Las actividades se determinan con los números de los nudos de sus extremos. Se clasifican por el programa en orden creciente de su nudo de comienzo. Se calculan primero los tiempos "más pronto" para cada nudo, después los "más tarde". Los tiempos más pronto y más tarde de cada actividad y los distintos márgenes se calculan cuando se imprimen los resultados a partir de los tiempos más pronto y más tarde de los nudos.

c) **Presentación de los datos**

Deben escribirse en el programa. Basta indicar para cada actividad:

- El número del nudo de comienzo,
- El número del nudo de final,
- Duración de la actividad.

La lista de datos debe acabarse con -1, -1, -1.

d) **Programa y hoja de resultados** (Pág. 118, 119)

La función TAB se empleó en las líneas 710 a 740 para obtener una mejor presentación de los resultados. Es una extensión del BASIC que se describe en el apéndice I.

PROGRAMA

```
100 PRINT "CAMINØ CRITICØ"
110 PRINT
120 DIM I (100), J (100), D (100), A (100), T (100), R (100)
130REM: LECTURA DE DATØS
140 FØR K1=1 TØ 300
150   READ I (K1), J (K1), D (K1)
160     IF I (K1) < 0 THEN 180
170 NEXT K1
180 LET N= K1 - 1
190REM: CLASIFICACIØN DE LAS ACTIVIDADES
200 FØR K1=1 TØ N
210   LET A (K1) = K1
220 NEXT K1
230 FØR K2=1 TØ N
240   LET K3=0
250   FØR K1=1 TØ N - K2
260     LET A1=A (K1)
270     LET A2=A (K1 + 1)
280     IF I (A1) < I (A2) THEN 340
290     IF I (A1) > I (A2) THEN 310
300     IF J (A1) < J (A2) THEN 340
310     LET K3= K3 + 1
320     LET A (K1) = A2
330     LET A (K1 + 1) = A1
340   NEXT K1
350   IF K3=0 THEN 380
360 NEXT K2
370REM: INICIALIZACIØN
380FØR K1=1 TØ N
390   LET T (K1) =0
400   LET R (K1) =0
410 NEXT K1
420REM: CALCULØ DE LØS TIEMPØS MAS PRØNTØ PARA LØS NUDØS
430 FØR K1=1 TØ N
440   LET I1 =I (A (K1))
450   LET J1 =J (A (K1))
460   LET M=T (I1) + D (A (K1))
470   IF T (J1) =0 THEN 490
480   IF T (J1) >= M THEN 500
490   LET T (J1) = M
500 NEXT K1
510REM: CALCULØ DE LØS TIEMPØS MAS TARDE PARA LØS NUDØS
520 LET R (J (A (N))) = T (J (A (N)))
530 FØR K1 = N TØ 1 STEP - 1
540   LET I1 = I (A (K1))
550   LET M=R (J (A (K1))) - D (A (K1))
560   IF T (I1) =0 THEN 580
570   IF R (I1) <=M THEN 590
580   LET R (I1) =M
590 NEXT K1
600REM: ESCRITURA
610 PRINT
620 PRINT " ACTIVIDAD      COMIENZO     ",
630 PRINT "     FIN            MARGENES "
640 PRINT " I   J    TIJ     MAS TARDE",
650 PRINT " MAS TARDE    PRONTO    LIBRE    INDEP"
660 PRINT
670 FØR K1=1 TØ N
680   LET A1 =A (K1)
690   LET I1 = I (A1)
700   LET J1 =J (A1)
```

(CONT.)

```
710    PRINT I1; TAB (3); J1; TAB (7); D (A1); TAB (15);
720    PRINT T (I1); TAB (22); R (J1) - D (A1); TAB (30);
730 PRINT T (I1) + D (A1); TAB (37); R (J1); TAB (45);
740    PRINT R (J1) - T (I1) - D (A1); TAB (52); T (J1) - T (I1) - D (A1); TAB (59);
750
760    PRINT T (J1) - R (I1) - D (A1); " ";
770    IF R (J1) - T (I1) <> D (A1) THEN 800
780    PRINT "ACT. CRIT."
700    GØ TØ 810
800    PRINT
810 NEXT K1
820 DATA 2, 3, 10, 3, 4, 16, 4, 5, 8, 6, 7, 8
830 DATA 1, 2, 8, 2, 4, 2, 7, 8, 2, 5, 7, 3
840 DATA 4, 6, 12, 3, 6, 7, 3, 5, 4, -1, -1, -1
850 END
```

HOJA DE RESULTADOS

```
CAMINØ CRITICØ
```

ACTIVIDAD			COMIENZO		FIN		MARGENES		
I	J	TIJ	MAS	TARDE	MAS	TARDE	PRONTO	LIBRE	INDEP
1	2	8	0	0	8	8	0	0	0 ACT. CRIT.
2	3	10	8	8	18	18	0	0	0 act. CRIT.
2	4	2	8	32	10	34	24	24	24
3	4	16	18	18	34	34	0	0	0 ACT. CRIT.
3	5	4	18	47	22	51	29	20	20
3	6	7	18	39	25	46	21	21	21
4	5	8	34	43	42	51	9	0	0
4	6	12	34	34	46	46	0	0	0 ACT. CRIT.
5	7	3	42	51	45	54	9	9	0
6	7	8	46	46	54	54	0	0	0 ACT. CRIT.
7	8	2	54	54	56	56	0	0	0 ACT. CRIT.

APENDICE I

Extensión del BASIC

Después que el lenguaje BASIC fue desarrollado en el Dartmouth College, ha sido ampliado con nuevas posibilidades por diferentes constructores. Aquí se presentan las principales. Se aconseja, sin embargo, verificar los detalles de su uso en el manual BASIC del servicio de Tiempo Compartido que se emplee.

a) **Operación con las matrices**

Las tablas pueden emplearse para representar matrices.

Ciertas instrucciones se han introducido para operar globalmente con matrices; todas ellas comienzan con MAT.

Lectura

100 MAT READ A (I, J) lee una matriz de I línea y J columnas. Antes se deben haber definido I y J y la tabla A debe haberse declarado con dimensiones iguales o superiores a I, J. Los datos se deben dar en el orden

A (1,1), ..., A (1, J), A (2,1), ..., A (2, J), ..., A (I, J)

Escritura

100 MAT PRINT X imprime la matriz X por líneas.

Cálculos

Si A, B y C son matrices, se pueden emplear las instrucciones siguientes:

```
100  MAT A = B + C
100  MAT A = B - C
100  MAT A = B * C
100  MAT A = INV (B)   (inversión)
100  MAT A = TRN (B)   (transposición)
100  MAT A = ZER (I,J) (la matriz A debe tener las dimensiones I, J, todos
                        sus elementos toman el valor 0)
100  MAT A = CØN (I,J) (la matriz A debe tener las dimensiones I, J, todos
                        sus elementos toman el valor 1)
100  MAT A = IDN (I,I) (la matriz A (I, I) se transforma en la matriz unidad).
```

Una matriz puede también multiplicarse por una expresión aritmética. Por ejemplo:

100 MAT A = (K ↑ 2 - X * Y) * B siendo A y B matrices y K, Y y X variables no indexadas.

b) **Variables con contenido alfanumérico**

Las variables usadas en la primera versión del BASIC no podían tomar más que valores numéricos. Una extensión del BASIC permite manipular variables que contienen tiras de caracteres (letras, cifras y otros). Estas variables se denominan con una letra seguida del signo $ (dólar), y se llaman variables con contenido alfanumérico.

A$, J$, W$...

Los ejemplos de instrucciones permitidas son:

- 100 LET K$ = "EXTENSIØN BASIC"

La variable K$ contendrá: "EXTENSIØ N BASIC". El número máximo de caracteres depende del sistema empleado y puede variar de 15 a 72.

- 110 LET I$ = K$

La variable I$ contendrá la misma tira de caracteres que K$: EXTENSION BASIC.

- 120 PRINT I$

Se imprime el contenido de I$.

- 100 READ X, B$, Y, C$, K$
  520 DATA 3, PEREZ, 7, "3, BIG STR.",
  "NY - 10011"

Las variables con contenido alfanumérico pueden leerse como las otras. B$, C$ y K$ contendrán respectivamente: PEREZ; 3, BIG STR.; NY-10011.

Los datos alfanuméricos deben ir entre comillas cuando:

- Contengan caracteres que tienen un sentido especial en BASIC (signos de puntuación, signos operativos, paréntesis...).
- El primer carácter es una cifra.

Por ejemplo en 950 DATA "3, BIG STR.", MIAMI no es necesario poner MIAMI entre comillas, por no tener más que letras. Por el contrario 3, BIG STR. al comenzar por una cifra y tener una coma, debe ir entre comillas.

```
100 INPUT B$, K, S$
```

permite leer variables alfanuméricas tecleadas por el usuario durante la ejecución.

```
100 IF L$ = "DURAN" THEN 300
110 IF "MARZO" < = S$ THEN 90
120 IF Y$ > T$ THEN 340
```

En estas pruebas, el signo > quiere decir: "después en el orden alfabético" si las variables no contienen más que caracteres alfabéticos. Si ellas contienen también caracteres numéricos, es necesario usar códigos para hacer la comparación.

Cuando se comparan dos variables de longitudes diferentes, la variable más corta se compara con la parte correspondiente de la otra.

Se pueden emplear listas con contenido alfanumérico debiéndose declarar:

```
100 DIM B (20,30), C$ (12), X$ (20), A (50)
```

Las tablas con contenido alfanumérico no se aceptan en general.

c) **Instrucción LET**

Un mismo valor puede asignarse a varias variables de la manera siguiente:

```
100 LET D = E = F = T(I) = SQR (B ↑ 2 - 4 * A * C)
```

d) **Instrucción DEF**

Es posible definir una función teniendo más de un argumento como en el ejemplo siguiente:

```
100  DEF FND(A,B,C) = SQR (B ↑ 2 - 4 * A * C)
```

e) **Instrucción ON**

Permite hacer pasar el control a diferentes instrucciones según el valor de una variable o expresión, sustituyendo así una secuencia de instrucciones IF. Por ejemplo:

```
100  ØN A + B GØ TØ 340, 210, 260
```

El control pasa a las líneas 340, 210 ó 260 cuando el mayor entero inferior a A + B es respectivamente igual a 1,2, 3. Si este entero es inferior a 1 ó superior a 3, se imprime un mensaje de error.

No hay límite en el número de números de líneas que siguen al GØ TØ.

f) **Función TAB**

Se obtiene una mejor presentación de los resultados empleando funciones TAB en una instrucción PRINT, como en el ejemplo siguiente:

```
100  PRINT X(I); TAB (10); Y(I); TAB (20); Z(I)
```

Las columnas de la hoja de resultados están numeradas de 0 a 74, el valor de X (I) comenzará a imprimirse en la columna 0, el de Y (I) en la columna 10 y el de Z (I) en la columna 20.

APENDICE II

A. Lenguaje de comandos de los sistemas HP 2000, B, C y F

a) **Identificación**

Antes de poner usar el sistema, el usuario debe "identificarse" de la manera siguiente (los caracteres escritos por el *ordenador* van subrayados):

```
45
PLEASE LØG IN
HELLØ-A895, EX2
READY
```

El usuario comienza por teclear un número cualquiera para indicar que quiere usar el sistema. El ordenador responde pidiendo al usuario que se identifique ("PLEASE LOG IN"). El usuario se identifica tecleando "HELLO-A895, EX2", siendo A895 el número de identificación y EX2 su clave. El sistema verifica que este número y clave son válidos y si lo son contesta al usuario que está preparado a recibir sus mensajes (READY).

b) **Entrada del programa**

Después que el ordenador haya escrito READY, el usuario puede escribir su programa directamente, línea a línea. Es posible corregir un error en una cierta línea volviendo a teclear la misma línea con el mismo número de línea, de la manera siguiente:

```
270  NEXT I
280  INPØT A,B      ← error
290  LET C = A * B
300  LET D = AB     ← error
280  INPUT A,B
300  LET D = A/B
```

Las instrucciones puestas en memoria son en este caso:

```
270  NEXT I
280  INPUT A,B
290  LET C = A * B
300  LET D = A/B
```

Las primeras líneas 280 y 300 tecleadas por el usuario son reemplazadas por las líneas que tienen los mismos números y tecleadas posteriormente.

c) **Salvar**

El usuario puede conservar en memoria auxiliar un programa, que acaba de escribir o modificar, tecleando:

```
NAME - PROG 1
SAVE
```

La primera línea permite atribuir un nombre al programa, lo que es necesario para poderlo invocar después.

La segunda línea da orden al ordenador de conservar el programa PRØG1.

d) **Utilización de un programa guardado en memoria auxiliar**

Un programa salvado con el comando SAVE debe ser llamado antes de poderlo usar (ejecutar, modificar, listar...). El programa PRØG1 se le llama tecleando GET-PRØG1. Luego puede usarse como si el usuario lo terminara de teclear. En particular, se puede ejecutar o modificar varias veces sin que sea necesario llamarle de nuevo cada vez.

e) Ejecución

Cuando el programa ha sido tecleado por el usuario, o llamado con el comando GET-, si estaba salvado de antes, se puede ejecutar con el comando RUN. Los resultados (o mensajes de error) se imprimen por el sistema. El sistema indica que la ejecución se acaba tecleando DØNE:

```
965
PLEASE LØG IN
HELLØ-A895, EX2
READY
GET-PRØG1
RUN
?
8,15.7,3.632
   ↕          Resultados para los datos anteriores
DØNE
RUN
?
8,15.7,3.632
   ↕          Resultados para los datos anteriores
DØNE
```

f) Borrado

Si el programa PRØG1 guardado con el comando SAVE ya no es útil en memoria auxiliar, puede ser borrado con el comando KILL-PRØG1.

g) Modificaciones

Es posible modificar un programa salvado y llamado con el comando GET- de la misma manera que un programa recién entrado por el teclado. Basta volver a teclear las líneas erróneas con su mismo número de línea pero corregidas y si es una línea a añadir con un número intermedio correspondiente a la posición de la línea en el programa.

En el caso de un programa antiguo y salvado, esto no modifica más que una copia del programa, quedando el programa original en memoria y protegido de toda modificación. Para modificar la versión del programa guardada en memoria (para poder llamar posteriormente la versión modificada del programa y no la versión original), es necesario borrar la versión anterior y luego salvar la nueva versión de la manera siguiente.

```
:
GET-PRØG1
100  INPUT X,Y                  }
367  LET D = (X + Y) * 2        } modificaciones
KILL-PRØG1 ←————————————  borrado de la antigua versión de PRØG1
SAVE ←——————————————————  salvar la nueva versión de PRØG1
```

Es necesario borrar la antigua versión de PRØG1, antes de salvar la nueva, ya que no puede haber en memoria dos programas con el mismo nombre. Si se quieren conservar las dos versiones de PRØG1, se puede por ejemplo dar otro nombre a la nueva versión. En este caso PRØG1 se modifica, la nueva versión se le da otro nombre y se salva de la manera siguiente:

```
:
GET-PRØG1
100  INPUT X,Y                  }
367  LET D = (X + Y) * 2        } modificaciones
NAME-PRØG2
SAVE
:
```

h) **Listado**

El listado completo de un programa se obtiene tecleando LIST. A veces es preferible listar sólo una parte del programa. Esto se obtiene tecleando por ejemplo LIST-60, 130 que lista las líneas del programa de 60 a 130 ambas inclusive.

i) **Parada**

Para acabar el funcionamiento del terminal, basta teclear BYE. El sistema indica el tiempo transcurrido desde que el usuario se identificó en el teletipo.

B. Resumen de Basic

RESUMEN DE LAS INSTRUCCIONES BASIC

Instrucciones	*Ejemplos*	*Páginas*
READ	READ K, A(4), L	35
DATA	DATA 0.54,432,153.5	35
INPUT	INPUT A,B,C(I*L), Z(5)	41
LET	LET A = (K*12) ↑ L1 + B(4)	34
GØ TØ	GØ TØ 250	38
PRINT	PRINT "VALOR ACTUALIZADO:"; V	37
REM	REM – – FACTURACIØN	34
STØP	STØP	54
END	END	37
IF... THEN...	IF K > 2 THEN 170	54
FØR... = ...TØ ...STEP...	FØR I = A + B TØ A↑2 STEP 4	59
NEXT	NEXT I	60
DEF	DEF FNA(X) = 0.01 * INT(X * 100 + 0,5)	77
GØ SUB	GØ SUB 250	81
RETURN	RETURN	81
DIM	DIM A(15,20), B(5)	70
Extensión BASIC		
MAT READ	MAT READ A(I,J), B(4,3), C	120
MAT PRINT	MAT PRINT X,Z	120
MAT	MAT A = B * C	121

ØN...GØ TØ...	ØN X↑2 GØ TØ 125,270,270	123
LET	LET A = X(4) = B2 = Z * T(4)↑2	122
DEF	DEF FND(A,B,C) = SQR(B↑2 – 4 * S * C)	123

RESUMEN DE FUNCIONES BASIC

Funciones	*Ejemplos*	*Páginas*
SIN	LET B2 = C * SIN(L1)	74
CØS	LET Ø1 = CØS(3.14 * K/180)	74
TAN	IF TAN(X) ⩾ 0.30 THEN 325	74
ATN	DEF FNA(Y) = 0.5 * ATN(Y)	74
EXP	LET K2 = EXP(A + B/2)	74
ABS	IF ABS(J) ⩾ 1 THEN 90	74
SQR	DEF FN1(M) = SQR(M/3) * K	74
LØG	PRINT "LØG DE"; L; "="; LØG(L)	74
INT	IF A3 = INT (A3) THEN 150	74
SNG	LET P = P/2 * SNG(V1 * V2)	74
RND	LET A1 = INT (RND(L) + 0.5)	74

Extensión BASIC

TAB	PRINT A(J); TAB(12); B(J); TAB(20); A(J)/B(J)	123
INV	MAT A = INV(B)	121
TRN	MAT A = TRN(B)	121
ZER	MAT A = ZER(9,15)	121
CØN	MAT A = CØN(12,14)	121
IDN	MAT A = IDN(10,10)	121

Otros libros sobre INFORMATICA publicados por

PARANINFO S.A.

Generalidades

ABRAMSON.– Teoría de la información y codificación. 5ª edición. 216 páginas.
FLORES.– Estructuración y proceso de datos. 3ª edición. 478 páginas.
GARCIA SANTESMASES.– Cibernética. Aspectos y tendencias actuales. 188 págs.
GOSLING.– Códigos para ordenadores y microprocesadores. 80 páginas.
KASATKIN.– El ABC de la Cibernética. 2ª edición. 152 páginas.
LOTT.– Elementos de proceso de datos. 310 páginas.
OLIVETTI.– Diccionario de informática. Inglés-español. 4ª edición. 272 páginas.
O'NEAL.– Sistemas electrónicos de proceso de datos. 436 páginas.
SCHMIDT y MEYERS.– Introducción a los ordenadores y al proceso de datos. 3ª edición. 452 páginas.
URMAIEV.– Calculadores analógicos. Elementos de simulación. 296 páginas.

Hardware (Equipo Físico)

ANGULO.– Electrónica digital moderna. 4ª edición. 680 páginas.
ANGULO.– Electrónica fundamental Teoría y práctica. Desde la válvula hasta el circuito integrado. 6 tomos.
T/6 – Teoría: Circuitos integrados digitales y analógicos. Hacia el microprocesador. Práctica: Montajes y experimentación con circuitos integrados lógicos y operacionales. 2ª edicición. 432 páginas.
ANGULO.– Memorias de burbujas magnéticas. 280 páginas.
ANGULO.– Microprocesadores. Arquitectura, programación y desarrollo de sistemas. 2ª edición. 628 páginas.
ANGULO.– Microprocesadores. Curso sobre aplicaciones en sistemas industriales. 3ª edición. 576 páginas.
ANGULO.– Microprocesadores. Fundamentos, diseño y aplicaciones en la industria y en los microprocesadores. 3ª edición. 764 páginas.
ANGULO.– Microprocesadores. Diseño práctico de sistemas. 424 páginas.
GARLAND.– Diseño de sistemas microprocesadores. 192 páginas.
HALSALL y LISTER.– Fundamentos de microprocesadores. 220 páginas.
ROBIN y MAURIN.– Interconexión de microprocesadores. 192 páginas.
SHELLEY.– Microelectrónica. 120 páginas.
WEBER.– Introducción a los métodos de la técnica digital. 196 páginas.

Lenguajes

CHECROUN.– BASIC. Programación de microordenadores. 2ª edición. 112 páginas.

GALAN PASCUAL.– Programación con el lenguaje COBOL. 328 páginas.

ROSSI.– BASIC. Curso acelerado. 2ª edición. 224 páginas.

SANCHIS LLORCA y MORALES LOZANO.– Programación con el lenguaje PASCAL. 4ª edición. 368 páginas.

Aplicaciones e informática profesional

ANGULO.– Prácticas de microelectrónica y microinformática. 260 páginas.

BANKS.– Microordenadores. ¿Cómo funcionan? ¿Para qué sirven? Introducción. Software. Aplicaciones en industria y comercio.

BELLIDO.– ZX81. Curso de programación BASIC. 2ª edición. 128 páginas.

BELLIDO.– Cómo programar su Spectrum. 2ª edición. 128 páginas.

BELLIDO.– Cómo usar los colores y los gráficos en el Spectrum. 96 páginas.

ESCUDERO.– (Centro de Investigación UAM-IBM).– Reconocimiento de patrones. Fundamentos teóricos, algorítmo y aplicaciones de la moderna técnica denominada "Pattern Recognition". 690 páginas.

GAUTHIER y PONTO.– Diseño de programas para sistemas. 2ª edición. 290 págs.

GILDERSLEEVE.– Las tablas de decisión y su aplicación al proceso de datos. Enseñanza programada. 208 páginas.

HARTMAN, MATTHES y PROEME.– Manual de los sistemas de información (ARDI). Análisis. Requisitos y su determinación. Diseño y desarrollo. Implantación y evaluación. 2 vols.

T/1 – Dirección del Proyecto. Estudio de factibilidad: diseño y análisis del sistema. Desarrollo del sistema. Puesta en marcha del sistema y evaluación. 5ª edición. 382 páginas.

T/2 – Técnicas y estándares. 5ª edición. 384 páginas.

PANNELL, JACKSON y LUCAS.– El microordenador en la pequeña empresa.

B A S I C
Introducción a la Programación

El ordenador ya no es una herramienta reservada a los especialistas. Un gran número de investigadores, funcionarios, ejecutivos y estudiantes saben cómo funciona un ordenador y lo que éste puede o no hacer.

El BASIC es un lenguaje de programación relativamente nuevo que ha tenido un gran éxito entre los usuarios no especialistas en informática. El BASIC es muy fácil de aprender y, por este mismo motivo, muy fácil de usar. Es el lenguaje ideal para realizar programas de tamaño y complejidad medios.

Este libro está destinado a los que desean escribir sus propios programas, así como a todos los que desean conocer cómo es posible dar instrucciones a un ordenador, y su texto está estructurado en cuatro partes: una introducción a la programación; la definición del lenguaje BASIC; cómo realizar un programa en BASIC; y programas útiles.

EDITORIAL PARANINFO S.A
Magallanes, 25 - Madrid-15

ISBN: 84-283-1215-X