basic

PROGRAMACION DE MICROORDENADORES

A. CHECROUN

LeG PARANINFO SA

BASIC
Programación de microordenadores

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

Título original francés:
BASIC. PROGRAMMATION DES MICROORDINATEURS

IMPRESO EN ESPAÑA
PRINTED IN SPAIN

ISBN: 2-04-012122-6 (edición francesa)
ISBN: 84-283-1244-3 (edición española)
Depósito legal: M. 20.379.—1985

PARANINFO S.A. Magallanes, 25 - MADRID (15) (3-3561)

ARTES GRÁFICAS BENZAL, S. A. Virtudes, 7. 28010 MADRID

Indice de materias

Prologo

La microinformática permite considerar aplicaciones que afectan a nuestra vida cotidiana tanto profesional como familiar.

El tamaño reducido de los ordenadores actuales, su seguridad de funcionamiento (es decir que precisan muy poco mantenimiento), su facilidad de uso y por fin su coste reducido ponen estas herramientas al alcance de todos.

Hemos querido explicar de la forma más sencilla posible este fenómeno técnico que, sin ninguna duda, provocará un cambio social importante y en breve plazo. Este libro se ha escrito para el lector que conoce poco o nada los ordenadores, pero que un día u otro tendrá que utilizarlos.

Después de un breve resumen histórico, definiremos los elementos constitutivos de un sistema informático construido alrededor de un microordenador, así como los papeles respectivos que juega cada elemento.

Describiremos de una forma simplificada el principio de funcionamiento de un ordenador, destacando los conceptos más importantes, en especial la noción de lenguaje de programación y los diferentes niveles de lenguajes posibles.

Actualmente, todos los microordenadores, e incluso muchos ordenadores clásicos se programan con Basic. El capítulo 2 está dedicado a la enseñanza de este lenguaje, en su forma más normal.

Los capítulos 3 y 4 serán útiles al lector iniciado que haya asimilado los conceptos preliminares y que quiera avanzar más en su conocimiento.

El capítulo 3 está dedicado a la descripción de sentencias más difíciles que se emplean en algunos sistemas. Para ello se ha elegido uno de los sistemas más extendidos: El Basic Plus de Digital, pero muchas de las sentencias son idénticas para otros constructores.

El capítulo 4 introduce la noción de fichero y la forma de su tratamiento con el Basic Plus.

Esta edición se ha revisado como resultado de las numerosas discusiones y críticas de colegas y colaboradores.

Estoy especialmente agradecido a Marc Podgrodski por su ayuda en los dos últimos capítulos, así como a Pierre Durieux y Antoine Termine que revisaron y corrigieron algunos ejercicios.

Bibliografía

J. ARSAC, *Les systèmes de conduite des ordinateurs,* Dunod.

J. P. BOUHOT *et al., Un fil d'Ariane,* Tomes I et II, Editions de l'Informatique.

B. DEL VALLEE, *Comment fonctionne un ordinateur; les systèmes d'exploitation,* Dunod.

J. C. LARRECHE, *Le Basic, une introduction à la programmation.* Eyrolles (Traducido al español con el título: *Basic. Introducción a la programación,* por Ed. Paraninfo, Madrid (España).

L. NOLIN, *Formalisation des notions de machine et de programme,* Dunod.

J. C. SIMON, *Introduction à la structure et au fonctionnement des ordinateurs,* Masson et Cie.

R. ZAKS, *Introduction aux microordinateurs individuels et professionnels,* SYBEX (1978).

CAPITULO 1

Los microordenadores

1.1 DESDE LOS ORDENADORES A LOS MICROORDENADORES

Como es bien sabido, un ordenador está formado por componentes electrónicos. Desde los años 1950 en los que se fabricaban los ordenadores electrónicos con válvulas, cuyo elemento principal era la Unidad Central (lógica y de cálculo) y que necesitaba locales muy climatizados y preparados, hasta nuestros días, en los que la unidad de proceso tiene un volumen insignificante y un precio pequeño frente a los demás elementos del sistema, los componente electrónicos han ido sufriendo una miniaturización progresiva.

No entraremos en el detalle de la tecnología empleada para obtener estas miniaturizaciones. Daremos sencillamente una idea del progreso realizado en este campo, citando los resultados obtenidos en los periodos principales que jalonan la evolución del concepto del ordenador.

Primeramente hay dos precursores:

- 1944, Aiken en Harvard realiza el "MARK I" que combinaba válvulas y relés. Esta máquina efectuaba una suma en 0,25 segundos y una multiplicación en 4 segundos.
- 1946, Eckert y Mauchly de la Universidad de Pensilvania, ponen a punto el primer ordenador completamente electrónico el ENIAC (Electronic Numerical Integrator and Automatic Calculator). Pesaba varias decenas de toneladas, consumía la energía eléctrica que precisaba un tren de gran tamaño y necesitaba una climatización importante. Su funcionamiento era vigilado por doscientas personas, tenía una avería cada diez minutos y realizaba cinco mil sumas en un segundo.

- 1948, después del estudio teórico fundamental de Von Neuman, aparece el EDVAC (Electronic Discrete Variable Automatic Calculator). Se va imponiendo la codificación binaria de la información.
- 1952, descripción del efecto de campo por Shockley.
- 1956 a 1968. Se reemplazan las válvulas por transistores. Aparecen las memorias de ferritas y los lenguajes de programación. Resulta con ello una mayor seguridad de funcionamiento, velocidad y flexibilidad de uso. Hacen su aparición los discos magnéticos y las impresoras.
- 1965 a 1970: Circuitos integrados, consolas de visualización. Métodos conversacionales. Redes y teleproceso. Las velocidades se multiplican por 100 y los precios se dividen por 10.
- 1970 a 1980: Progreso considerable en la miniaturización de los componentes y circuitos, pudiéndose tener un ordenador en una caja de zapatos. Las velocidades se siguen aumentando y de nuevo los precios se dividen por 10.

Actualmente , se puede encontrar en el mercado una unidad central de 32.000 caracteres (32 Kc) provista de una pantalla de rayos catódicos, de un teclado, de dos discos flexibles de 180.000 caracteres, de una impresora, todo alrededor de 5.000 dólares y que ocupa un volumen equivalente a 3 cajones de oficina.

Lo anterior es posible con los microordenadores cuya unidad central está hecha alrededor de un circuito microprocesador en una sola pieza, con componentes LSI (Large Scale Integration). Sobre una pastilla de Silicio de algunos milímetros cuadrados se pueden disponer hasta 50.000 transistores y sus conexiones.

1.2 ESTRUCTURA DE UN MICROORDENADOR

Como todo ordenador, un microordenador debe realizar tres funciones principales:

- una función de proceso o cálculo
- una función de almacenamiento
- una función de intercambio.

Dichas tres funciones las cumplen respectivamente los tres elementos siguientes que son básicos en la estructura de un ordenador:

- *La unidad central* (U.C.): se encarga de ejecutar los tratamientos indicados por las "instrucciones" que previamente ha encontrado en su "memoria".
- *La memoria central* (M.C.): empleada para almacenar los *programas* (o conjuntos de instrucciones) y los *datos*.
- *Los órganos de entrada-salida* (E/S): permiten la comunicación con el exterior. Por ejemplo un teclado permite la entrada de datos, una impresora sirve para la salida de resultados.

Resumamos pues las principales definiciones esquematizando los cuatro elementos funcionales de todo ordenador:

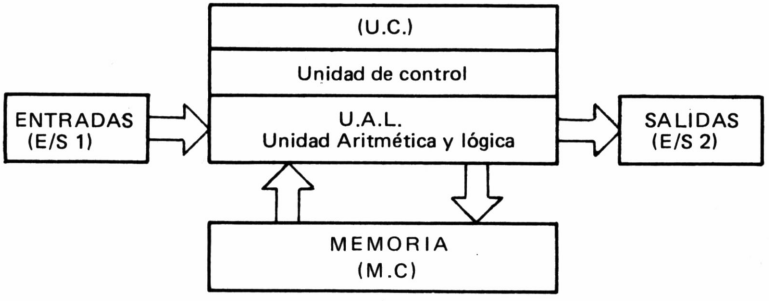


Fig. 1

Físicamente, un microordenador está formado por dos o tres tarjetas lógicas y un dispositivo de alimentación para la U.C. (que incluye la M. C.). Los periféricos de E/S más normales son un teclado y una pantalla de rayos catódicos. El teclado (similar al de una máquina de escribir) permitirá la entrada de caracteres (un carácter por tecla) codificados en binario. La pantalla representará los caracteres engendrados en salida por el microordenador.

Por supuesto, se podrán conectar periféricos de tipos variados, según los usos que se deseen. Por ejemplo, en Salida, vendrá bien muy a menudo una impresora para guardar resultados intermedios, escribir cartas comerciales ... En Entrada, se comprende fácilmente la utilidad de órganos que permitan la lectura rápida de gran cantidad de información; para ello son adecuadas las unidades de minidiscos ("floppy") o de cintas magnéticas (minicassettes).

Los periféricos mencionados se pueden conectar al microordenador mediante un BUS, que es una vía de comunicación entre la unidad central y los diferentes órganos que se le conecten.

Un BUS es pues un conjunto de líneas agrupadas por funciones. Se pueden hallar normalmente:

- Un BUS de datos
- Un BUS de direcciones
- Un BUS de control.

1.3 PRINCIPIO DE FUNCIONAMIENTO

1.3.1 Representación de la información

Recordemos que el microordenador es una máquina destinada a manejar informaciones de diversos tipos.

Estas informaciones podrán ser:

- *de datos:* elementos que definen el problema a resolver. Un fichero es un grupo de datos de la misma naturaleza;
- *de instrucciones:* indicaciones sobre los procesos a efectuar. Un programa es una secuencia de instrucciones que realizan un cierto proceso.

En todos los casos estas informaciones se efectuarán en el interior de la máquina con una representación binaria; lo que se puede considerar una limitación debida a la naturaleza física de los soportes empleados.

En el diseño, el constructor elige una cierta estructura de información que se corresponde con los circuitos cableados y/o microprogramados correspondientes. De esta forma se definirá el conjunto de las operaciones elementales posibles y el volumen global disponible.

La unidad de información es el bit (es una contracción de "Binary Digit"), pero cada máquina se caracteriza por la menor cantidad de información accesible directamente de una vez: a este conjunto de bits se le llama *palabra.* La mayor parte de los microordenadores tienen una palabra de 8 bits, pero también los hay con palabras de 16 bits y pronto los habrá con palabras de 32 bits.

Con palabras de 8 bits se podrá disponer de un *vocabulario* potencial de $2^8 = 256$ palabras distintas. Se aprecia fácilmente el interés de tener máquinas con palabras de 32 bits, suponiendo que no hubiera limitaciones de tamaño o coste.

Una vez fijada la longitud de las palabras, el constructor define luego la estructura de una instrucción. Por ejemplo, se tiene en un caso muy favorable:

código de operación	dirección del primer operando	dirección del segundo operando	dirección del resultado
8 bits	8 bits	8 bits	8 bits

es decir se tiene una instrucción de tres direcciones que necesita 32 bits.

Con un código de operación de 8 bits, se podrá disponer como máximo de 256 operaciones diferentes, lo que es más que suficiente.

El conjunto de los códigos de operación constituye el *vocabulario de la máquina.* Por ejemplo:

00010100	significará Sumar.

Por otra parte el constructor define la capacidad de almacenamiento de su máquina, pensando en la estructura física subyacente. En general, la memoria del ordenador podrá contener varios millares de palabras. Debido a esto, se ha definido una unidad para medir la capacidad, que corresponde a 1024 palabras y se representa con K. Se dirá por ejemplo, que se tiene una memoria de 32 Kpalabras (caso muy normal). Los microordenadores actuales pueden tener memorias de 64 hasta 128 Kpalabras en sus versiones comerciales.

1.3.2 Organización de la memoria

Si se esquematiza la memoria con un rectángulo que contenga casillas numeradas, se pondrán tantas casillas como palabras tenga dicha memoria. Por ejemplo 32768 casillas para una memoria de 32 Kpalabras.

Una parte de esta memoria contendrá los programas grabados previamente por el constructor, en especial el *monitor* o programa capaz de interpretar las instrucciones del usuario. Otra parte de la memoria contendrá el programa del usuario.

Es decir, la memoria permitirá almacenar todos los datos necesarios.

Como se ha dicho anteriormente, todas las informaciones "memorizadas" lo están en forma de *palabras* (por ejemplo de 8 bits, llamándose entonces octetos o también "bytes"); a cada palabra le corresponde un

número o dirección. En nuestro ejemplo, la primera casilla lleva la dirección 0, la última lleva la dirección 255 (ver figura 2). En binario estas direcciones se escriben:

00000000 y 11111111

(que es el número máximo que se puede escribir con 8 bits).

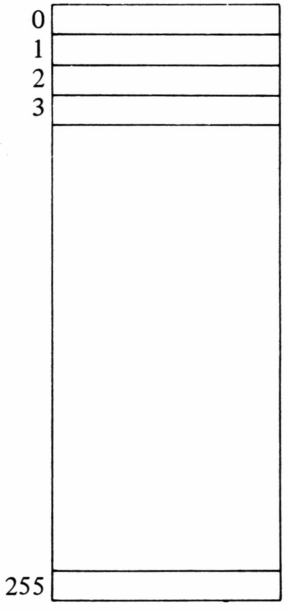


Fig. 2

Notas. Las tecnologías empleadas en los microordenadores son variadas y el usuario se encontrará al principio con muchas siglas, de las que quizás las más importantes son las dos siguientes:

- las R.O.M. (Read Only Memory) o memorias fijas, destinadas a almacenar los programas que no se vayan a modificar, como ocurre por ejemplo con el monitor o sistema operativo;
- las R.A.M. (Random Access Memory), o memorias de acceso al azar, y que pueden ser leídas o escritas, pero tienen el inconveniente de ser volátiles, o sea que se pierde su contenido si se desconecta la tensión de alimentación. El usuario deberá pues guardar la información útil en otro soporte, antes de desconectar la máquina.

1.3.3 Ejemplo de funcionamiento

Para comprender el principio del funcionamiento de un ordenador, vamos a describir una configuración muy simplificada, pero que tiene los elementos esenciales para tratar la información. Dicha configuración es la siguiente:

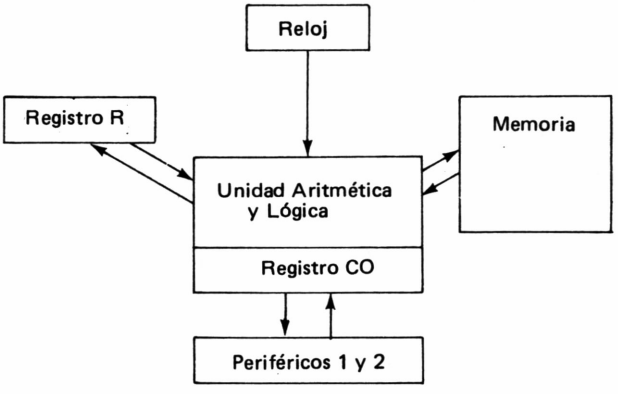
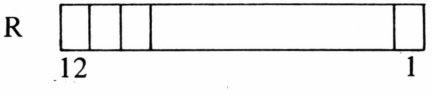


Fig. 3

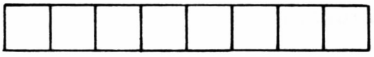
– *El reloj:* es un contador electrónico que envía impulsos regularmente espaciados, permitiendo así sincronizar las acciones de la U.C.

– *La memoria:* es un conjunto de 256 palabras de 12 bits; cada palabra lleva una dirección (de 0 a 255).

– *El registro* R: es un grupo de elementos electrónicos que pueden tomar uno de dos estados estables que se denominan 0 y 1 (básculas electrónicas). Hay un total de 12 básculas en este registro, con lo que puede copiarse enteramente una palabra de memoria.



– *El registro C.O. o contador ordinal:* es un grupo de 8 básculas del mismo tipo. Permite pues representar números binarios que lleguen hasta 11111111 o sea 255; contiene la dirección de la palabra a dirigirse en la memoria.

CO 

– *Los periféricos 1 y 2:* son por ejemplo un teclado que permite introducir información y una máquina de escribir que permite imprimir los resultados.
– *La unidad de cálculo (U.C.):* bajo la acción del reloj y siguiendo un programa de instrucciones grabado previamente, la U.C. puede acceder:

  – al estado de una palabra de la memoria
  – al estado de R
  – al estado del C.O.
  – a toda información que provenga del periférico de entrada.

  la UC puede pues modificar el contenido de R y de CO, y también ordenar la impresión de caracteres en el periférico de Salida.

Hipótesis de funcionamiento

Al conectar la máquina, se realiza automáticamente un cierto número de acciones elementales:

Acción 1: el contenido del CO toma valor 32
Acción 2: lectura del contenido del CO
Acción 3: lectura de la palabra cuya dirección viene dada por el CO
Acción 4: realización de un proceso que depende del contenido de la palabra de memoria que se acaba de leer
Acción 5: el contenido del CO se aumenta en una unidad y se vuelve a la acción 1

Por otra parte, el constructor ha definido también el conjunto de instrucciones posibles que se resumen en el cuadro de la Fig. 4.

Se observará que cada instrucción de la figura 4 está formada por un código de operación de 4 bits y una dirección de 8 bits.

Para simplificar la explicación se ha empleado el convenio siguiente:

n representa la dirección y *(n)* el contenido de la palabra de dirección *n*.

La flecha → significa "poner en".

Contenido de la casilla apuntada por el CO	Acción resultante
1010 aaaaaaaa	Grabar en el registro R el contenido de la palabra de memoria de dirección A (notación simbólica: (A) → R)
1011 aaaaaaaa	Grabar en la palabra de memoria de dirección A el contenido de R (R) → A
1000 aaaaaaaa	Restar el contenido de R del contenido de la palabra de memoria de dirección A y grabar el resultado en A (A) − (R) → A
1101 aaaaaaaa	Sumar el contenido de R al contenido de la palabra de memoria de dirección A y grabar el resultado en A (A) + (R) → A
1100 aaaaaaaa	Restar 1 del contenido de la palabra de memoria de dirección A y grabar el resultado en A (A) − 1 → A
0001 aaaaaaaa	Grabación de la cantidad A en CO A → CO
0101 aaaaaaaa	Grabación de la cantidad A en CO en el caso en que la casilla de dirección 1 contenga 0, si no, no hacer nada (1) = 0 { sí: A → CO ; no: nada
0111 00000001	Salida del número contenido en el registro R por el periférico nº 1
0110 00000010	Espera que se teclee una cifra en el periférico nº 2 y entonces pone en R el código binario de dicha cifra
0000 00000000	Parada de la máquina (se corta la comunicación entre el reloj y la Unidad de Cálculo)

Fig. 4

Supongamos que hayamos grabado en la memoria la serie siguiente de instrucciones:

Dirección	Contenido de la palabra de memoria
16	0000 0000 0000
32	0110 0000 0010
33	1011 0000 0001
34	1010 0001 0000
35	1011 0000 0010
36	0101 0010 1000
37	0110 0000 0010
38	1101 0000 0010
39	1100 0000 0001
40	0001 0010 0011
41	1010 0000 0010
42	0111 0000 0001
43	0000 0000 0000

Fig. 5

Si ahora se teclean en el teclado los números siguientes:

2, 4 y 5

se verá aparecer el número 9 impreso en el papel de la impresora.

Análisis del funcionamiento

(Ver cuadro de la página 19, figura 6).

Teniendo el cuadro o programa de la figura 5 en memoria, si el usuario hubiera tecleado 3, 2, 1, 4, el ordenador habría impreso el valor 7. De una manera general, el ordenador calcula la suma de un número cualquiera de cifras, siendo la primera cifra *n* tecleada el número de cifras a leer y sumar. La máquina imprime el resultado cuando se han sumado *n* cifras. El cuadro binario de la figura 5 constituye el programa de la máquina; la operación de grabar el programa en la máquina es programar el ordenador.

En efecto, la secuencia de acciones anteriores se traduce así:

– pone en la casilla 1 (contador) el valor *n*
– pone a cero la casilla 2 (totalizador)

Nº de casilla indicado por	Acciones	Efectos sobre el contenido de las casillas 16, 1 y 2 y de los registros R y CO				
		C.O.	16	1	2	R
32	Espera del 1er carácter tecleado (es un 2)					
	2 → R	32	0	?	?	2
33	(R) → 1	33	0	2	?	2
34	(16) → R	34	0	2	?	0
35	(R) → 2	35	0	2	0	0
36	Prueba de (1): (1) ≠ 0 Ninguna acción	36	0	2	0	0
37	Espera de un carácter (es un 4)					
	4 → R	37	0	2	0	4
38	(R) + (2) → 2	38	0	2	4	4
39	(1) − 1 → 1	39	0	1	4	4
40	35 → C.O.	35	0	1	4	4
36	Prueba de (1): (1) ≠ 0 Ninguna acción	36	0	1	4	4
37	Espera de un carácter (es un 5)					
	5 → R	37	0	1	4	5
38	(R) + (2) → 2	38	0	1	9	5
39	(1) − 1 → 1	39	0	0	9	5
40	35 → C.O.	35	0	0	9	5
36	Prueba de (1): (1) = 0 luego					
	40 → C.O.	40	0	0	9	5
41	(2) → R	41	0	0	9	9
42	Salida de R: el teletipo teclea el carácter 9	42	0	0	9	9
43	Parada del programa					

Fig. 6

– mientras el contador no sea cero, lectura de la cifra siguiente, sumándola al totalizador (operación de la casilla 38), restando 1 del contenido del contador.

– impresión del contenido del totalizador (casilla 2) cuando el contador (casilla 1) tiene cero (se han sumado las *n* cifras).

1.4 LOS DIFERENTES NIVELES DE LOS LENGUAJES

1.4.1 El lenguaje máquina

Como hemos tenido ocasión de decir anteriormente, todo ordenador posee un vocabulario elemental definido por su constructor y que se representa con algunas decenas de palabras binarias que corresponden al conjunto de acciones que puede realizar la máquina. Combinando estas palabras según unas reglas dadas por el constructor, se forma un programa.

Por ejemplo, la figura 5 es un programa en lenguaje máquina (característico de la máquina simplificada descrita antes).

Cada máquina tiene su propio lenguaje, luego un programa escrito para una de ellas no podrá servir para otra.

1.4.2 Lenguajes ensambladores

Es evidente que no es nada práctico el manejo humano de un lenguaje máquina. Para hacer el lenguaje máquina menos árido y más nemotécnico se ha pensado que:

- cada código de operación
- cada caso de trabajo
- cada variable

se representen por grupos de caracteres que formen un "código nemotécnico". Se tendrá por ejemplo:

Código binario	Significación	Código Nemotécnico
1010	LEER	(A) →R
1011	ESCR	(R) →A
1000	REST	(A) − (R) →A
1101	SUMA	(R) + (A) →A
1100	DISM	(A) − 1 →A
0001	SALT	Salto incondicional
0101	SALC	Salto si cero
0111	SALI	SALIDA
0110	ENTR	ENTRADA
0000	FIN	PARADA

Fig. 7

y el programa de la figura 5 quedará así:

1ª columna	2ª columna	3ª columna	
	PRØGR	SUMA	Declaraciones
	VARIA		
CØNT			Datos
TØTAL			
	CØNST		
CERØ		0	Constantes
	ØPER		Operaciones
	ENTR	2	
	ESCR	CØNT	
	LEER	CERØ	
	ESCR	TØTAL	
BUCLE	SALC	FINALG	
	ENTR	2	
	SUMA	TØTAL	
	DISM	CØNT	
	SALT	BUCLE	
FINALG	LEER	TØTAL	
	SALI	1	
	FIN		
	END		

Fig. 8

Este programa se ha presentado como un fichero. Representa lo que se introducirá en la máquina y que se traducirá por el *ensamblador*. Este

ensamblador es un programa ya colocado en memoria y que traducirá este fichero al lenguaje binario y que en nuestro caso concreto resultará el programa descrito anteriormente.

El ensamblador analizará la secuencia de caracteres del fichero de la página 21 de la manera siguiente:

- VARIA: asignación de las casillas 1, 2, etc., a los grupos de caracteres especificados en el párrafo VARIA.
- CØNST: asignación de las casillas 16, 17, etc., a los grupos de caracteres especificados en el párrafo CONST y colocación de los valores de estas constantes (dadas en la tercera columna) en las casillas correspondientes.
- ØPER: colocación del programa propiamente dicho, ya en binario, comenzando en la casilla 32; lo que supone para cada línea (o instrucción:
  - Si hay caracteres en la primera columna y siguientes (es decir distintos de blanco); asignar a estos caracteres la dirección de la casilla de memoria en la que el ensamblador coloca el código binario de la operación correspondiente a esta línea.
  - Traducir los caracteres de la segunda columna en su codificación binaria de la operación correspondiente y poner dicha traducción al principio de la palabra memoria (bits 12 a 9).
  - Traducir los caracteres de la tercera columna como sigue:
    * Si son cifras, traducir todo el número a binario y ponerlo en los bits 8 a 1.
    * Si no mirar si este grupo de caracteres corresponde al nombre de una variable, de una constante o de una etiqueta de instrucción (que aparecerá en la primera columna), traduciendo la dirección correspondiente a código binario y ponerla en los bits 8 a 1.
- END: parar el ensamblaje, se ha terminado el fichero o programa.

En el ejemplo precedente se ha presentado un lenguaje ensamblador y las acciones correspondientes al programa traductor respectivo.

Con este simbolismo se aprecia un progreso en la comprensión del programa por el usuario del ordenador. Los códigos de operación escritos de forma nemotécnica (LEER, SUMA, etc) y los nombres de los datos (variables, constantes o direcciones del programa) son mucho más inteligibles.

1.4.3 Lenguajes evolucionados

Se quiere hacer el programa aún más fácil de manejar escribiéndolo en un lenguaje que recuerde lo más posible al lenguaje natural del programador (inglés, castellano, ...).

El lenguaje Basic es un ejemplo de estos lenguajes; traduciendo a Basic el programa de los ejemplos anteriores, se obtiene:

```
10  INPUT C
20  LET   T = 0
30  FØR   I = 1    TØ  C
40  INPUT N
50  LET   T = T + N
60  NEXT  I
70  PRINT T
80  STØP
90  END
```

es decir:

10 Entrada de C (Contador).
20 Se pone a cero T (Total).
30 Hacer I = 1 antes de comenzar el bucle y si I > C se sale del bucle yendo al número 70.
40 Entrada de una cifra N.
50 Se suma N a T.
60 Se incrementa I en uno y se salta al comienzo del bucle, a la prueba de I > C.
70 Salida de T (Total).
80 Código de fin.
90 Código de fin de fichero.

Para obtener la traducción en lenguaje máquina de un programa similar al anterior es necesario tener en la memoria de la máquina un programa traductor, denominado *compilador,* que analiza las instrucciones una a una, señalando los posibles errores de sintaxis mediante mensajes de error y coloca en memoria el programa binario correspondiente.

CAPITULO 2

El lenguaje Basic

Con el fin de hacer más cómoda la presentación de este capítulo, hemos adoptado un camino en espiral. Es decir que partimos de ejemplos elementales y vamos progresando en profundidad hacia ejemplos de complejidad creciente, con lo que estaremos obligados a repasar varias veces las mismas nociones.

2.1 LAS PRIMERAS NOCIONES DEL LENGUAJE

Instrucciones, Programa, Variable, Dato.

Teclear sobre el teclado

```
PRINT 23-9
```

dando luego a la tecla RETURN. Se obtiene sobre la pantalla (o en la impresora)

```
14
READY
```

Habíamos escrito una *instrucción* (o *sentencia* de Basic) que ha sido interpretada y ejecutada por el compilador tan pronto como se ha dado a la tecla RETURN. Es una instrucción denominada *sentencia directa;* que nos muestra la notable ventaja del Basic de ser un lenguaje "conversacional": cada vez que se de a la tecla RETURN, se considera como una instrucción al conjunto de caracteres tecleados, se interpreta y si es correcta (no tiene errores), se ejecuta la instrucción.

Se hará así cada vez que sólo se necesite ejecutar una instrucción. Pero los problemas más sencillos necesitarán varias instrucciones, pudiendo en los casos más complejos tener millares de instrucciones. El conjunto de instrucciones necesarias para la resolución de un problema dado constituirá un *programa.* Veamos un ejemplo sencillo:

PROBLEMA 1 .

Calcular el cuadrado y el cubo de los primeros números enteros

Programa 1:

```
10  READ N
20  LET C = N * N
30  LET K = C * N
40  PRINT N, C, K
50  GOTO 10
60  DATA 1, 2, 3, 4, 5, 6, 7, 8, 9
70  END
RUN
```

Comentarios:

– Este programa tiene en total siete líneas numeradas de 10 en 10. Cada línea representa una instrucción Basic y los números de las líneas sirven para precisar el orden en el que se debe ejecutar las instrucciones (se puede llegar hasta 99999). Así se habría podido escribir, sin cambiar el programa:

```
10  READ N
60  DATA 1, 2, 3, 4, 5, 6, 7, 8, 9
40  PRINT N, C, K
20  LET C = N * N
```

Evidentemente, se pueden numerar las líneas en el orden secuencial natural. Sin embargo, la numeración de 10 en 10 permite una mayor flexibilidad, debido a que se podrán añadir en todo momento instrucciones olvidadas o modificaciones de los programas; por ejemplo:

```
65 DATA 10, 11, 12, . . . , 19
```

nos permite añadir una serie de números además de los que ya tenía el programa.

- Cada instrucción comienza con una *orden* que es una palabra o una expresión inglesa indicando el tipo de acción que se va a ejecutar.
- Los caracteres que siguen a esta *orden* son los símbolos que representan los *datos*, las *variables* o los *operadores*.

Así en la instrucción o sentencia

```
30 LET K = C * N
```

30 es el número de la sentencia
LET es la orden Basic
K, C y N son las variables
= y * son los operadores aritméticos.

- No se tienen en cuenta los espacios entre los diferentes caracteres de una línea: el compilador los ignora, pero son útiles para facilitar la lectura del programa. Se puede escribir por ejemplo:

```
10 READN
20 LETC = N*N
```

Entre los cinco tipos de sentencias que figuran en el ejemplo anterior, algunas de ellas merecen algunos comentarios adicionales.

La *orden* READ (instrucción 10), que es una orden de lectura, debe ir siempre acompañada de instrucciones DATA, que de hecho son ficheros de datos creados desde la compilación y consultados o leídos secuencialmente, es decir que al ejecutarse la instrucción READ N, el calculador tomará el primer valor del fichero y lo asignará al símbolo N. En la continuación del programa se tendrá pues N = 1, hasta que se ejecute una nueva orden READ o una instrucción LET N = (ver después).

La orden LET (instrucciones 20 y 30) permite efectuar operaciones aritméticas y poner el resultado en la casilla de memoria cuya dirección está representada por el símbolo situado a la izquierda del signo "=". Por ejemplo: 20 realiza el producto (*) de N por N y asigna el resultado a la variable C. Por supuesto, no se podrá tener más que una variable a la izquierda del signo igual.

Veamos aparecer aquí la noción de "variable", es decir la utilización de símbolos que son interpretados por el compilador como direcciones en la memoria.

Es sobre el contenido de estas direcciones sobre el que se aplican todas las operaciones previstas en el programa. En Basic se representa una varia-

ble con una sola letra o con una letra seguida de un dígito, por ejemplo: N, X, R2, Y5.

Los operandos aritméticos normales se representan en Basic con los símbolos siguientes:

	Paréntesis	( )
	Elevación a una potencia	↑
(mismo peso)	cociente	/
	producto	*
(mismo peso)	diferencia	−
	suma	+

Los operadores se han dado en orden decreciente de ponderación. Es decir que en una expresión aritmética dada se calculan primero los paréntesis más internos. En el interior de los paréntesis se efectúan primero las potencias, después los cocientes y productos y por último las adiciones y diferencias. Si se tienen dos operadores de la misma jerarquía, se comienza por el de más a la izquierda.

Ejemplo: expresar en Basic:

$$h = \frac{\dfrac{a+b}{c-d}}{e.f} \cdot (x+y)^3$$

```
20 LET H = ( (A + B)/( (C − D)/(E − F) ) ) * (X + Y) ↑ 3
```

Uno se puede asombrar del número de paréntesis empleados en Basic con relación a los que se necesitan en la expresión dada. Esto se deriva del hecho de que en el primer caso se emplea una representación en dos dimensiones (es decir que la posición de los símbolos en el plano nos indica el orden en que se deben aplicar los operadores). En cambio en Basic la escritura debe tener un caracter unidimensional que se pasará al compilador. Se han añadido flechas para emparejar los paréntesis, aunque no hay posibilidad de confusión si se va asociando cada paréntesis de cierre con el último paréntesis de apertura. Una vez ejecutado el contenido de este paréntesis, se borra el paréntesis y se sigue ejecutando con este sistema. El compilador usa un algoritmo de este tipo para traducir una expresión, es decir para descomponer la instrucción Basic en instrucciones binarias de máquina.

Notas

- Es frecuente que se puedan ahorrar paréntesis gracias al convenio de la prioridad de las operaciones. En el ejemplo anterior se podría escribir:

```
20  LET H = (A + B) * E * F/(C - D) * (X + Y) ↑ 3
```

- Se puede reducir de esta forma el número de operaciones a realizar en una expresión, modificando sencillamente el orden en que aparecen los operadores (si es posible).
- Conviene comprobar siempre que hay el mismo número de paréntesis de apertura que de cierre.
- Repetimos que el valor del resultado de la expresión aritmética situada a la derecha del signo = se asigna a la variable que aparece a la izquierda. Es legal una expresión del tipo:

```
50 LET I = I + N
```

  lo que hace es aumentar la variable I en N unidades; si I tenía el valor 30 y N vale 6, después de la ejecución de la instrucción 50, I valdrá 36.
- La mayor parte de los compiladores de Basic admiten el uso de las instrucciones aritméticas llevando o no la orden LET. En lo sucesivo se escribirá por ejemplo:

```
50 I = I + N
```

  sobreentendiéndose la orden LET.

La *orden* PRINT (instrucción 40) no haría falta comentarla, permite imprimir los valores tomados por las variables N, C y K en un instante dado. Más adelante veremos cómo colocar los resultados dentro de la página impresa.

La *orden* GOTO (instrucción 50) nos manda a la línea 10, es decir a leer el dato siguiente, con lo que N irá tomando sucesivamente los valores 2, 3, ..., 9; permitiendo la repetición del cálculo anterior con valores diferentes. Se llama bifurcación incondicional, permite realizar bucles, o sea repetir una misma parte del programa. Más adelante, veremos otras formas de realizar bucles de programa.

- El calculador se para cuando no encuentra datos en el fichero DATA (veremos después que es más seguro controlar el fin de un programa que no esperar a que se vacíe el fichero de datos).
- Hemos visto que cada orden READ manda la lectura de un dato del fichero DATA en el orden en que aparecen. Es importante verificar la correspondencia entre las órdenes de lectura y la sucesión de datos en DATA. Sin embargo, en lugar de escribir

```
60  DATA  1, 2, 3, 4, ..., 50
```

se habría podido escribir también:

```
60  DATA 1, 2, 3, 4, 5
61  DATA     6, 7, 8, 9, 10
---------------
69  DATA 46, 47, 48, 49, 50
```

La orden END indica al compilador que es la última instrucción del programa. Por consiguiente, todo programa Basic deberá llevar END como última instrucción.

El hecho de apretar la tecla RETURN después de la última instrucción del programa 70 END no hace nada. Para conseguir la ejecución del programa hay que teclear la orden RUN (orden del monitor, que no lleva pues número de línea). Así, contrariamente a las órdenes directas, las instrucciones de un programa son interpretadas a medida que se teclean (con RETURN), pero no se ejecutan todas ellas como un programa hasta que se teclea RUN. Los resultados de la ejecución son:

1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	452
9	81	729

```
? OUT OF DATA ERROR IN 10
READY
```

La *orden* GOTO *n* es una orden de bifurcación incondicional. Cuando se llega a esta instrucción, el programa continúa en la instrucción *n*, que puede estar antes o después de donde estamos ahora.

En nuestro ejemplo, la instrucción GOTO 10 nos manda a la línea 10, es decir a la lectura del dato siguiente en el fichero DATA. Cuando éste último se ha leído por completo, el programa sigue queriendo leer aún más datos, con lo que se justifica el mensaje de error que aparece después de los resultados.

2.2 ENTRADA DE DATOS Y SALIDA DE RESULTADOS

Utilización de las órdenes INPUT, READ, PRINT y DATA.

PROBLEMA 2

Se quiere crear un fichero en el que se escriben los pedidos de N clientes. Los datos de cada cliente son los siguientes:

– número de pedido P (número entero de 3 cifras)
– cantidad pedida C (puede llegar hasta 10.000)
– precio unitario U (5 cifras)

Cuando se ha creado el fichero se quiere representar su contenido en la pantalla, cada línea es un pedido con su número de pedido y sus C y U correspondientes.

Programa 2:

```
 10  INPUT N
 20  DATA 1, 2450, 3030, 2, 5600, 1150, 3, 256, 724, 4,
 .....
 .....
 50  DATA 100, 2314, 1810
 60  READ P, C, U
 90  PRINT P, C, U
 95  GOTO 60
110  END
```

Comentarios:

– La orden INPUT es una instrucción de entrada de datos en modo conversacional. Cuando se encuentra esta orden, el programa se in-

terrumpe para pedir al usuario que introduzca el valor correspondiente a la variable N. En la pantalla aparece un signo de interrogación; cuando se teclea el valor, continua la ejecución del programa. Es posible introducir los valores de varias variables con una sola instrucción INPUT, separándolas con una coma:

```
10 INPUT A, B, C
```

si al ejecutarse, al aparecer el signo de interrogación, se teclean 10, 30, 80 resultará que se asigna

el valor 10 a la variable A
el valor 30 a la variable B
el valor 80 a la variable C

En nuestro ejemplo, si se quieren considerar 100 pedidos, se tecleará 100 cuando aparezca el signo de interrogación.

– La formación del fichero se reduce a introducir por el teclado los datos sucesivos correspondientes a los números de pedido, de la cantidad a pedir y del precio unitario.

DATA es la orden que permite crear tal fichero.

Obsérvese que cada valor que sigue a la orden DATA *va separado del siguiente por una coma;* no se pone coma después del último número de una línea, salvo si el número de valores a introducir sobrepasa la capacidad de una línea, en cuyo caso se sigue en las líneas siguientes.

Ejemplo:

```
100  DATA 25, 4870, 1385, 25684300, 290125
144, 294, 3249, 57856
100  READ A, B, C, D, E, F, G, H
```

tendría como resultado el poner

```
      25  en A
    4870  en B
    1385  en C
25684300  en D
290125144 en E
     294  en F
    3249  en G
   57856  en H
```

Una instrucción DATA no es ejecutable; debe ir siempre acompañada de una instrucción READ.

– La *orden* READ desencadena la lectura del fichero DATA en el orden secuencial de las informaciones que contiene. En nuestro ejemplo, los tres primeros valores encontrados se asignarán a las variables P, C y U. Se tendrá pues en primer lugar en la memoria:

P = 1 C = 2450 U = 3030

La instrucción GOTO 60 va a provocar enseguida una nueva lectura de 3 números consecutivos y así sucesivamente hasta agotarse al fichero. Por supuesto aquí también habrá que separar las variables con comas.

La instrucción PRINT P, C, U efectuará la impresión en una línea de los valores que tengan en ese momento las variables P, C, U, o sea:

1 2450 3030

De esta forma se pueden escribir hasta 5 resultados en una misma línea (son los límites de la pantalla que está dividida en 5 zonas, con la utilización de las comas entre las variables). Se puede llegar hasta 8 resultados en una línea, usando el ";" en lugar de ",".

La instrucción 95 GOTO 60 provoca, como se ha visto con el READ, la impresión repetida de 3 números sucesivos.

En definitiva, se verá aparecer sobre la pantalla la tabla siguiente

```
 1          2450        3030
 2          5600        1150
 3           256         724
 4           '           '
 '           '           '
 '           '           '
100         2314        1810
? OUT OF DATA ERROR IN 60
```

Volvemos a encontrar el mensaje de error que indica que el fichero se ha agotado, pero que el programa pedía más datos.

2.3 LAS INSTRUCCIONES DE PRUEBA O BIFURCACIONES CONDICIONALES

PROBLEMA 2 bis

A partir del fichero definido en el problema nº 2, calcular la suma S = C + U e imprimir los datos relativos a aquellos pedidos tales que S sea inferior a 3000.

Programa 2 bis:

```
 10  INPUT N
 20  DATA .....
 .....
 50  DATA .....
 55  PRINT "PEDIDOS DIMINUTOS"
 57  I = 1
 60  READ P, C, U
 65  I = I + 1
 70  S = C + U
 80  IF S >= 3000 THEN 95
 90  PRINT P, C, U
 95  IF I < N THEN 60
110  END
```

Comentarios:

Para responder a la cuestión planteada hemos tenido que añadir las instrucciones 55, 57, 65, 70, 80 y además se ha cambiado la instrucción 95. Esta última 95 GOTO 60 era una instrucción de bifurcación incondicional. Ahora vamos a ver las *bifurcaciones condicionales.*

Al ejecutarse la instrucción

```
80 IF S >= 3000 THEN 95
```

la máquina compara el valor calculado de S con el número 3000. Si el valor de S es superior o igual a 3000 entonces la próxima instrucción que se ejecutará será la instrucción 95, en caso contrario (o sea si S < 3000) la máquina irá a ejecutar la instrucción que sigue en secuencia, o sea la:

```
90 PRINT P, C, U
```

Observemos a este respecto que el orden secuencial corresponde, como se dijo anteriormente, al orden creciente de los números de línea. Así, cada vez que S sea menor que 3000 se imprimirán en una línea los valores correspondientes de P, C y U.

En la línea 95 se realiza una nueva prueba para que el proceso se detenga al final de un número determinado de *bucles*. Para contar estos bucles se han empleado las instrucciones:

```
57 I = 1                  (inicialización de un contador)
65 I = I + 1              (incremento del contador)
95 IF I < N THEN 60       (prueba de parada)
```

En efecto cuando I = N se pasará a la siguiente instrucción en secuencia, es decir

```
110 END
```

La forma general de una instrucción IF es:

Nº de línea	IF	Condición a comprobar	THEN	Nº de línea adonde se bifurcará si se cumple la condición

La condición a comprobar tiene 3 elementos:
un "sujeto", una "relación" y un "objeto"

Hay 6 relaciones posibles

=	igual
>	mayor
<	menor
>=	mayor o igual
<=	menor o igual
<>	diferente.

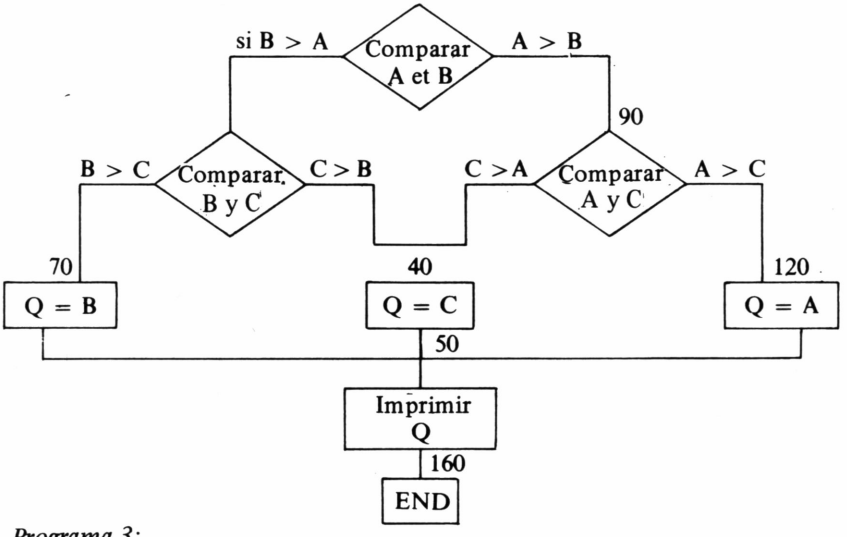
El sujeto y el objeto pueden ser variables, expresiones o números. Por ejemplo:

```
A5 > ((B + C)/K) * (M + P)
S * (Q + P) ↑ 2 = 15.2
124 < Y
MAR < > MAX
```

Se pueden suceder varias instrucciones IF en un programa. Supongamos el problema siguiente:

PROBLEMA 3

Se quiere determinar el mayor de los 3 números positivos A, B, C siguiendo el siguiente organigrama.



Programa 3:

```
 10  READ A, B, C
 20  IFA > B THEN 90
 30  IF B > C THEN 70
 40  Q = C
 50  PRINT Q
 60  GOTO 160
 70  Q = B
 80  GOTO 50
 90  IFA > C THEN 120
100  GOTO 40
120  Q = A
130  GOTO 50
150  DATA 3, 1, 4
160  END
```

Repaso de la orden PRINT

Se recordará que el Programa 2 bis llevaba la instrucción 55 PRINT "PEDIDOS DIMINUTOS", que permite la salida por la pantalla de comentarios sobre los resultados y de una manera general de cualquier texto que se desee. Para ello basta poner el texto entre comillas. Se podrá obtener así el cuadro siguiente:

– LISTA DE PEDIDOS –

P	C	U
1	2450	3020
2	5600	1150
'	'	'
'	'	'

empleando las instrucciones:

```
55  PRINT "LISTA DE PEDIDOS"
56  PRINT "...P......C.......U......."
```

La orden PRINT sola provoca el salto de una línea. Se podrán igualmente mezclar impresiones de textos y de resultados en una misma orden PRINT. Por ejemplo

```
70 PRINT "C =", C, "X=", X, "Y=", Y
```

2.4 LOS BUCLES DE UN PROGRAMA

El problema no 2bis podría admitir el programa Basic siguiente

Programa 2ter:

```
20  DATA ..............
.............................
55  PRINT "PEDIDOS DIMINUTOS"
57  FOR I = 1 TO N
60  READ P, C, U
70  S = C + U
80  IF S > = 3000 THEN 95
90  PRINT P, C, U, S
95  NEXT I
110 END
```

La instrucción 57 FOR I = 1 TO N quiere decir: "ejecutar todas las instrucciones que siguen hasta la orden NEXT inclusive, N veces". Ella realiza pues el bucle de programa completo definido en el párrafo anterior: el contador se crea, incrementa y prueba, de una forma implícita. La forma general de esta instrucción es:

```
 20  FOR I = I1 TO I2 STEP S
     "
     "
     "
     "
130  NEXT I
```

Que se traducirá literalmente así: "Para el índice I yendo de I1 a I2 con un incremento de S, ejecutar las instrucciones que siguen hasta encontrar la orden NEXT inclusive. Observemos que el índice utilizado en el FOR debe repetirse en NEXT. Los índices pueden ser enteros o reales, positivos, negativos o nulos. También pueden reemplazarse por una expresión aritmética o por variables.

Ejemplos:

```
 25  FOR K = 10 TO 15
 34  FOR N5 = 50 TO 1 STEP - 1
 48  FOR J = 3.5 TO 7.5 STEP 0.35
104  FOR T = P TO L STEP J
134  FOR Z = B 3 TO (C + D)/2 STEP 5 * J
```

Naturalmente los valores de los símbolos o variables deberán haberse definido fuera del bucle, ya sea mediante órdenes READ o INPUT o bien con algún cálculo.

2.5 LAS VARIABLES CON SUBINDICES

PROBLEMA 4

Volver a tomar el fichero definido en el problema nº 2 y leer las M fichas que lo componen. Determinar para cada pedido la suma S = C + U y extraer los pedidos tales que S < 3000, calculando el total de los mismos.

Programa 4

```
10  DATA 1, 25, 645, 2, 368, 552, 3, 465, 2250
11  DATA 4, 236, 1248, 5, 695, 4250, 6, 182, 749
12  DATA 7, 707, 961, 8, 747, 4650, 9, 95, 1942
13  DATA 10, 951, 2687, 11, 433, 1044, 12, 285, 700
14  INPUT M
15  DIM N(M), C(M), U(M), S(M)
20  FOR I = 1 TO M
30  READ N(I), C(I), U(I)
40  NEXT I
50  PRINT " PEDIDOS INFERIORES A 3000"
60  PRINT
70  PRINT " -N---------C----------U---------S "
80  PRINT : PRINT
90  T = 0
100 FOR I = 1 TO M
110 S(I) = C(I) + U(I)
120 IF S(I) > = 3 000 THEN 140
130 PRINT N(I), C(I), U(I), S(I)
135 T = T + S(I)
140 NEXT I
145 PRINT
150 PRINT ≪TOTAL =≫, T
160 END
```

Al revés de lo que se hizo anteriormente, se ha querido esta vez leer el conjunto de datos del fichero, antes de comenzar el proceso. Así se gana en velocidad de ejecución. Para esto se debe tener de antemano la memoria necesaria; esto es lo que realiza la orden DIM (abreviatura de DIMENSION).

Cuando se escribe

```
14 DIM N(12), C(12), U(12), S(12)
```

se reservan bloques de 12 palabras de memoria cada uno (para N, C, U y S).

Los símbolos N, C, U, S designan cada uno una zona de memoria y para localizar un lugar particular dentro de la zona se emplea un índice.

En nuestro ejemplo, cuando se encuentra la instrucción

```
30 READ N(I), C(I), U(I)
```

el primer dato leído se almacenará en la palabra N(1), el segundo en la palabra C(1), el tercero en la U(1). El cuarto en N(2), ... Los símbolos N(I), C(I) ..., son variables con subíndices. Si el índice no sobrepasa a 10 la orden DIM es innecesaria.

Observemos que el índice M en la orden DIM puede ser una variable. Es decir se tiene

```
15 DIM N(M), C(M), U(M), S(M)
```

si se toma la precaución de definir la variable M anteriormente, como se ha hecho con la instrucción

```
14 INPUT M
```

que provocará la impresión de un ? y esperará a que se teclee un valor.

Esta solución tiene la ventaja de permitir una mayor flexibilidad en la utilización del programa. En efecto cuando se han terminado de teclear las órdenes DATA (fichero de datos), se sabe el número de los datos, pudiéndose entonces fijar M.

En nuestro ejemplo, bastará teclear 12 cuando aparezca el primer signo de interrogación en la pantalla.

Todas las operaciones válidas para las variables normales lo son también para las variables indexadas. El índice debe ser un entero positivo o nulo y debe haberse calculado en una instrucción anterior. Si el cálculo del índice diera un valor decimal, sólo se retendría la parte entera.

Por ejemplo si

```
30 K = 5/2
40 P(K) = 13.5
```

el valor 15.3 se pondrá en P(2).

– Retorno a las órdenes de impresión:

En este ejemplo se puede comprobar que con la ayuda de las comillas, la colocación de los datos dentro de la página es bastante fácil.

La instrucción 70 imprimirá N, C, U, S como títulos de las columnas.

– Nótese que se pueden escribir varias instrucciones en una misma línea si no se sobrepasa su capacidad. Para ello basta separar las instrucciones con ":"

Ejemplo

```
80 PRINT : PRINT : PRINT
```

equivale a

```
80  PRINT
81  PRINT
82  PRINT ——→ salto de 3 líneas
```

Ejecución del programa 4

```
RUN
? 12
PEDIDOS INFERIORES A 3000
```

N	C	U	S
1	25	645	670
2	368	552	920
3	465	2250	2717
4	236	1248	1484
6	182	729	931
7	707	961	1668
9	95	1842	1937
11	433	1044	1477
12	285	700	985

```
TOTAL = 12787
READY
```

Los índices pueden ser también expresiones aritméticas. Ejemplo:

```
50 P((K - 3)/4) = 160
```

que no será válida más que si K > 3.

2.6 BUCLES MULTIPLES. VARIABLES CON VARIOS SUBINDICES

PROBLEMA 5

Se quiere calcular los coeficientes C_n^k *del desarrollo del binomio de Newton.*

Se recuerda que $(a + b)^n = \sum_{k=1}^{n} C_n^k a^k b^{n-k}$ *siendo* $C_n^k = \frac{n!}{k!(n-k)!}$.

Se recuerda también la relación recurrente:

$$C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$$

Programa 5:

```
10  INPUT N
20  DIM C(N, N + 1)
30  C(1, 1) = 1
40  C(1, 2) = 1
50  FOR I = 2 TO N
60  C(I, 1) = 1
65  C(I, I + 1) = 1
70  FOR J = 2 TO I
75  C(I, J) = C(I - 1, J) + C(I - 1, J - 1)
77  NEXT J
80  NEXT I
90  FOR I = 1 TO N
100 FOR J = 1 TO I + 1
110 PRINT C(I, J);
120 NEXT J
125 PRINT
130 NEXT I
140 STOP
150 END
```

Comentarios:

– Hemos empleado el algoritmo del triángulo de Pascal: si se representan los coeficientes de los desarrollos de $(a + b)$, $(a + b)^2$, $(a + b)^3$, ..., $(a + b)^n$ y ocupando una línea para cada binomio se tiene

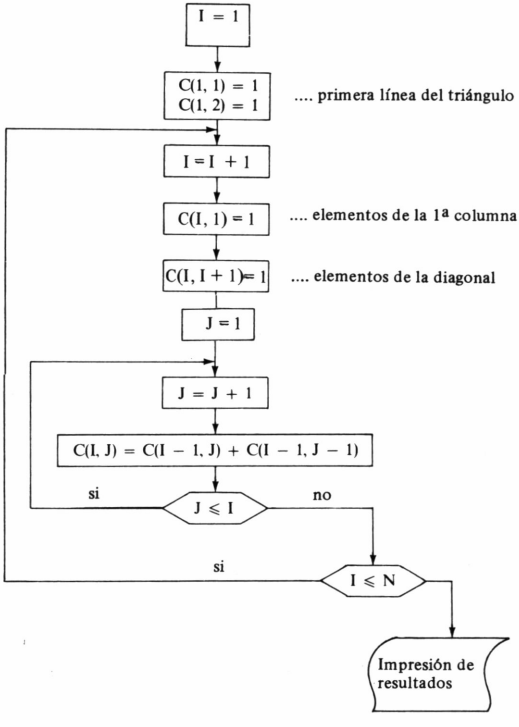
		$j=1$	$j=2$	$j=3$		$j=n$
para $i=1$	$a+b \rightarrow$	1	1			
para $i=2$	$(a+b)^2 \rightarrow$	1	2	1		
para $i=3$	$(a+b)^3 \rightarrow$	1	3	3	1	
.....						
para $i=n$	$(a+b)^n \rightarrow$	1	C_n^2	C_n^3	C_n^4	$C_n^n = 1$

dispuestos de esta forma, los coeficientes forman una matriz triangular cuyos elementos cumplen la relación de recurrencia:

$$C(i, j) = C(i - 1, j) + C(i - 1, j - 1)$$

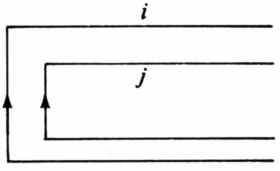
con las observaciones siguientes:

- los elementos de la 1ª columna son todos iguales a 1
- lo mismo para con los elementos de la diagonal de lo que se deduce el organigrama:

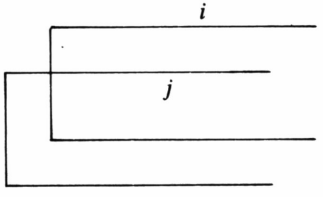


Volvamos al programa Basic:

- La línea 20 DIM C(N, N + 1) supone una reserva de N x (N + 1) palabras de memoria en las que se almacenarán los valores de los coeficientes calculados. Es evidente que también aquí N deberá definirse en una instrucción anterior: es lo que hace la instrucción 10.
- El cálculo de los coeficientes se realiza en las instrucciones 30 a 80. Este cálculo utiliza dos *bucles* FOR anidados, es decir un bucle con el índice J que se ejecuta para cada valor sucesivo del índice I, como se indica en este esquema:



Nota importante: no se puede entrar en un bucle más que por su principio. No se permiten solapamientos como éste



Por ejemplo la secuencia de instrucciones:

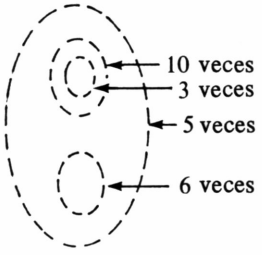
```
200  FOR I = 1 TO 8
210  FOR J = 1 TO 5
220  NEXT I
230  NEXT J
```

es inválida.

- Los bucles de programa constituyen la herramienta más importante de la programación ya que en la repetición de una misma secuencia de cálculos un gran número de veces, es donde el ordenador encuentra toda su potencia.

Ejercicios

1. Escribir la forma general de las instrucciones FOR y NEXT que realizan el esquema siguiente:



2. Calcular los k primeros números primos.
3. Realizar el producto de la matriz A (25, 25) por el vector B (25).
4. Hacer el producto de dos matrices rectangulares A (25,20), B (20, 30).

2.7 REPETICION DE UNA MISMA INSTRUCCION. FUNCIONES DE INSTRUCCION

2.7.1 La función DEF FN

Ocurre a menudo que un mismo cálculo que ocupa una instrucción aparece en diferentes sitios de un mismo programa. Entonces puede ser cómodo el disponer de una instrucción que se pueda llamar a voluntad.

Para esto es necesario *definir* esta función y luego poderla llamar.

- La definición se realiza con la orden DEF FN.
- La llamada se realiza con una instrucción de asignación típica.

Ejemplos:

Se trata de hacer la conversión de grados a radianes. Se puede escribir por ejemplo:

```
50  DEF FNR(D) = D * 3.14159/180
```

Cada vez que se desee hacer este tipo de conversión en el mismo programa, bastará llamar FNR (. . .):

Por ejemplo, escribiendo:

```
          70  T1 = FNR(37)
o bien    80  T2 = FNR(A + 5)
          ....................
o bien    90  PRINT FNR(14)
```

La forma general de la instrucción de definición de función es:

no línea DEF nombre (lista de argumentos) =
= expresión que comprende, entre otros, a estos argumentos

El nombre tiene que tener obligatoriamente 3 letras, cuyas dos primeras son FN. No se podrán definir más de 26 funciones que serán FNA, FNB, . . ., FNZ.

La expresión a la derecha del signo = puede ser cualquier expresión aritmética que esté contenida en una línea.

– Se pueden definir de la misma forma funciones de varias variables, como la

```
100  DEF FNV(X, Y, Z) = X ↑ 2 + Y ↑ 2 + Z ↑ 2
```

que se llamará, como es lógico, con 3 argumentos:

```
120  U1 = FNV(4, – 3, 2)
130  U2 = FNV(2.5, 1, 0)
```

2.7.2 Las funciones de biblioteca

Para realizar los cálculos clásicos se dispone también de funciones suministradas por una biblioteca permanente. Por ejemplo, a continuación se incluye una lista que encontraremos en todas las máquinas, con algunos cambios de nombre

SIN(X)	calcula el seno del ángulo X expresado en radianes
COS(X)	calcula el coseno del ángulo X expresado en radianes
TAN(X)	calcula la tangente del ángulo X expresado en radianes
ATN(X)	calcula el arco tangente expresado en radianes
EXP(X)	calcula la exponencial de X
ABS(X)	valor absoluto de X
SQR(X)	raíz cuadrada de X

INT(X)	parte entera de X
RND(X)	da un número aleatorio uniforme

Todas estas funciones tienen la misma forma general:

un nombre de 3 letras seguido de un argumento

El argumento puede ser una constante, una variable o una expresión. Se escribe por ejemplo:

```
20 A = SIN(B + C)
30 R = SQR(X2 + Y2)
40 P = ABS(X - Y) + SQR(X*Y+0.5)
```

Una función de biblioteca puede figurar en cualquier instrucción de cálculo o de impresión. En particular, se puede definir una instrucción de función que comprende una o varias funciones de biblioteca.

Ejemplo:

```
50 DEF FNW (A, B, C) = SQR (A ↑ 2 + B ↑ 2 + C ↑ 2)
```

– Veamos la función particular TAB(N), que permite posicionar los caracteres a representar o imprimir (TAB es una abreviatura de TABULADO). El parámetro N puede ser una constante, una variable o una expresión: en todos los casos se toma la parte entera del valor de la expresión. Si se escribe PRINT TAB (4.3) A, entonces el valor de A se presentará después de 4 espacios.

Se puede escribir también `PRINT TAB (10 - FN(X)); "0"`
o así `PRINT TAB(Y); "*"; TAB(Z); "0"`
esta última instrucción representa las curvas (Y) y (Z).

Ejercicios:

1. Dibujar las funciones $Y = \sin\left(\frac{x}{50} + \frac{3x}{1\,000}\right)$ et $Z = \cos\left(\frac{x}{30} + 0{,}25\right)$

   en un mismo gráfico.

2. Realizar una instrucción de función que permita alinear en columnas varios números decimales cualesquiera, con relación a la coma.

Por ejemplo si se quieren representar 347,50 y 1253,839 se quiere obtener:

```
345,50
1253,839
```

2.8 REPETICION DE UN MISMO PROGRAMA. NOCION DE SUBPROGRAMA

La utilización de una instrucción de función se limita al caso en el que todos los cálculos necesarios caben en una línea o instrucción Basic. Muy a menudo se presentan largas secuencias de cálculo que sería cómodo emplear en varios sitios de un programa. Así ocurre en el ejemplo siguiente:

PROBLEMA 6

Sea el fichero de facturas definido en los problemas 2, 3 y 4.

Determinemos de nuevo la suma S = C + U y clasifiquemos los pedidos en tres categorías

- *pedidos diminutos si $S \leqslant S1$*
- *pedidos medianos si $S1 < S \leqslant S2$*
- *pedidos grandes si $S > S2$*

Representados sucesivamente en sus tres clases con totales parciales y un total general.

Programa 6:

```
 5  DATA
10  INPUT M
20  DIM N(M), C(M), U(M), S(M)
30  FOR I = 1 TO M
40  READ N(I), C(I), U(I)
50  NEXT I
60  INPUT S1, S2
70  FOR I = 1 TO M
80  S(I) = C(I) + U(I)
85  NEXT I
86  T = 0
90  FOR K = 1 TO M
95  IF S(K) > S1 THEN 140
96  T = T + S(K)
```

```
110  PRINT « ...N.....C.....U.....S »
120  PRINT
130  PRINT N(K); C(K); U(K); S(K)
140  NEXT K
150  PRINT « TOTAL = ..... » , T
160  T = 0
165  FOR K = 1 TO M
167  IF (S(K) − S1) * (S2-S(K)) < 0 THEN 200
170  T = T + S(K)
180  PRINT « ...N.....C.....U.....S »
190  PRINT N(K); C(K); U(K); S(K)
185  PRINT
200  NEXT K
203  PRINT "TOTAL ....." , T
205  T = 0
210  FOR K = 1 TO M
220  IF S(K) − S2 < 0 THEN 270
230  T = T + S(K)
240  PRINT «...N.....C..... U.....S »
250  PRINT
260  PRINT N(K); C(K); U(K); S(K)
270  NEXT K
275  PRINT "TOTAL = ....." , T
280  STOP
999  END
```

Programa 6 bis:

```
 5  DATA.....
.....
10  INPUT M
20  DIM N(M), C(M), U(M), S(M)
30  FOR I = 1 TO M
40  READ N(I), C(I), U(I)
45  S(I) = X(I) + Y(I)
50  NEXT I
60  FOR J = 1 TO 3
70  INPUT S1, S2
80  GOSUB 100
90  NEXT J
95  STOP
```

```
100  T = 0
102  PRINT « ...N.....C.....U.....S »
105  FOR K = 1 TO M
110  IF (S(K) - S1) * (S2 - S(K)) < 0 THEN 130
115  T = T + S(K)
120  PRINT N(K); C(K); U(K); S(K)
130  NEXT K : PRINT
140  PRINT « TOTAL = ..... » , T
150  RETURN
999  END
```

Comentarios:

Los programas 6 y 6 bis hacen lo mismo. El programa 6 bis tiene 16 instrucciones Basic de menos, ya que las secuencias (86 a 150), (160 a 203) y (205 a 275) representan las mismas operaciones o por lo menos un proceso similar que puede hacerse sólo una vez con las instrucciones de la secuencia (100 a 140) del programa 6 bis.

Se ha decidido considerar esta última secuencia como un *subprograma* es decir un conjunto de instrucciones que realizan un tratamiento completo y susceptible de ser llamado en cualquier momento por el programa *principal.*

En el programa 6 bis, la secuencia (10 a 95) representa el *programa principal.* Se leen los datos del fichero DATA, se realizan las sumas S(I), después se llama 3 veces a la secuencia de tratamiento que va de 100 a 150.

Para utilizar un subprograma, es necesario poderlo llamar, poderle especificar el valor de los parámetros que necesita y cuando el tratamiento acabe, tiene que poder regresar al programa principal. Por eso se ha escrito

```
80 GOSUB 100
.......
150 RETURN
```

lo que quiere decir: ir a ejecutar las instrucciones que comienzan en la 100 hasta que se encuentre un RETURN. En ese momento se ejecutará la instrucción que sigue inmediatamente a la 80, que en nuestro caso será 90 NEXT J, es decir que en realidad hará

```
70 INPUT S1, S2
```

2.9 SUBPROGRAMAS INDEPENDIENTES (1)

Como se ha visto, la llamada GOSUB permite usar varias veces el mismo subprograma en un programa dado. Pero esta secuencia de instrucciones deberá formar parte del programa principal. O sea que el programa principal y los subprogramas se compilarán al mismo tiempo.

Existe otro tipo de subprogramas que permite ser compilado y almacenado en memoria independientemente del programa principal.

La forma general de un subprograma de este tipo sería:

Nombre (de 3 caracteres alfanuméricos)

10 "
20 "
• "
• "
• "

nº de línea RETURN

Su llamada se realiza en el programa principal con:

nº de línea CALL nombre del subprograma

Revisemos el ejemplo nº 6bis, que con esta nueva noción quedará así:

Programa 6ter:

```
 10  DATA.....
 20  INPUT M
 30  DIM N(M), C(M), U(M), S(M)
 40  FOR I = 1 TO M
 50  READ N(I), C(I), U(I)
 60  S(I) = C(I) + U(I)
 70  NEXT I
 80  FOR J = 1 TO 3
 90  INPUT S1, S2
 95  CALL CLA
 97  NEXT J
999  END
```

PROGRAMA PRINCIPAL

(1) Los párrafos que se marcan de esta forma se refieren a opciones Basic que no suelen estar disponibles en todos los sistemas.

Subprograma:

```
CLA
10  T = 0
20  FOR K = 1 TO M
30  IF (S(K) - S1) * (S2 - S(K)) < 0 THEN 60
40  T = T + S(K)
50  PRINT N(K); C(K); U(K); S(K)
15  PRINT « N.....C.....U.....S »
60  NEXT K : PRINT
70  PRINT « TOTAL = ..... », T
80  RETURN
```

Comentarios al programa 6ter:

La instrucción GOSUB llamaba a un subprograma incluido en el programa principal y compilado junto con él. La instrucción CALL permite por el contrario llamar a un subprograma que no forma parte del programa principal; es decir que el subprograma puede haberse compilado separadamente y guardado en memoria con un nombre determinado. Luego puede ser llamado desde diferentes programas.

En nuestro ejemplo la instrucción

```
95 CALL CLA
```

llama al subprograma CLA que usa las variables N, C, U, S cuyos valores se tenían en el programa principal en el momento de la llamada.

La vuelta al programa principal se hace aquí también con la instrucción RETURN (puede haber varias). El retorno se hace siempre a la primera instrucción que sigue a la instrucción CALL.

Naturalmente los números de las líneas del programa y del subprograma son independientes.

2.10 TRATAMIENTO DE LOS CARACTERES ALFANUMERICOS. LAS TIRAS DE CARACTERES

El Basic permite realizar ciertos tratamientos sobre variables no numéricas, es decir letras o caracteres especiales. A esas variables se las llama variables alfanuméricas.

Para indicar que una variable es alfanumérica se pone el signo $ detrás del nombre.

Se aceptan todas las operaciones de lectura escritura o asignación con estas variables.

Se podrá escribir por ejemplo:

```
20 LET A$ = "GRACIAS"
30 LET B$ = "MUCHAS"
40 PRINT B$; A$
50 END
```

este programa imprimirá el texto:

```
MUCHAS GRACIAS
```

El segundo miembro de una instrucción de asignación puede ser una tira de caracteres (en cuyo caso se pone entre comillas), como ocurre en el ejemplo anterior, o también una variable alfanumérica.

Esta variable contendrá ella misma una tira de caracteres, que no podrá exceder a 15 caracteres.

Ejemplo: 60 LET C$ = H$

Una variable alfanumérica puede tener subíndices. Entonces es necesario definirla con una instrucción DIM. Sólo se permite un subíndice.

Ejemplo: 10 DIM A$ (20)
que reservará 20 veces 15 caracteres en memoria.

– Lectura y escritura de tiras de caracteres: READ e INPUT pueden llevar variables alfanuméricas al mismo tiempo que variables numéricas.

Por ejemplo se puede escribir:

```
10 READ X$, Y$, M, P
20 INPUT N$, N
30 DATA "VIERNES", "13 MARZO 1982", 10, 15
```

observemos cómo los datos alfanuméricos se escriben entre comillas, que en realidad sólo son imprescindibles cuando la tira de caracteres comienza con una cifra o aparece alguna comilla.

En caso de duda se pueden poner siempre las comillas.

La instrucción PRINT se utiliza igual que con las variables numéricas.

– Comparaciones y pruebas con variables alfanuméricas.

Se puede usar la instrucción IF . . . THEN . . . : que compara dos tiras de caracteres uno a uno empleando su codificación BCD (decimal codificado en binario). Se ignoran los blancos.

Ejemplos:

```
100  IF A$ = « SMITH » THEN 240
120  IF B$ < > C$ THEN 300
410  IF « APRIL » < = M$ THEN 500
```

CAPITULO 3

Extensiones del Basic

Los lenguajes Basic que se emplean en los miniordenadores ofrecen posibilidades mucho más importantes que el Basic *elemental.*

En lo que sigue pasaremos revista a los aspectos más interesantes del Basic Plus de DEC empleados en los ordenadores PDP de la serie 11 y que funcionan con el sistema operativo RSTS (Sistema de tiempo compartido).

3.1 LAS VARIABLES

En el Basic Plus existen tres tipos de variables:

- variables reales
- variables enteras (1)
- variables de tira de caracteres.

En funcionamiento standard (modo implícito NO EXTEND) el nombre de cualquiera de estas tres variables está formado por uno o dos caracteres siendo el primero obligatoriamente una letra y el segundo (opcional) una cifra. La distinción entre los tres tipos de variables se hace por el signo que sigue al nombre: los nombres de las variables enteras van seguidas del signo %, mientras que las variables de tira de caracteres van seguidas del signo $.

Ejemplo:

```
10    A$ = "ABC"     \ Z3$ = "DEF"    \ X1$ = "GHI"
20    A%= 10%        \ Z3%= 5         \ X1%= 3%
```

(1) Como los enteros se almacenan en 2 octetos, sólo se pueden almacenar los enteros comprendidos entre los valores -32768 a 32767 ambos inclusive.

```
30      A =10.234      \ Z3 =2          \↑ X1 =-2.342
40      PRINT A$, Z3$,   X1$
\       PRINT A%, Z3%,   X1%
\       PRINT A , Z3 ,   X1
32767   END

Ready

RUNNH
ABC              DEF              GHI
 10               5                3
 10.234           2               -2.342

Ready
```

También se puede usar el modo EXTEND en cuyo caso los nombres de las variables pueden tener hasta 30 caracteres. Para indicar al intérprete el uso de este modo, es necesario ponerlo en la primera instrucción del programa.

Ejemplo:

```
5       EXTEND
10      VARIABLE.ENTERA%=5%
20      REAL = 123.23
30      TIRA DE CARACTERES$ = "ABCDEFG"
40      PRINT VARIABLE.ENTERA% , REAL , TIRA DE CARACTERES$

Ready

RUNNH
  5              123.23           ABCDEFG

Ready
```

3.2 OPERACIONES CON TIRAS DE CARACTERES

Una tira de caracteres está formada por una sucesión de caracteres alfanuméricos (definidos con el código ASCII) y considerada como un todo

Ejemplo:

```
10      A$ = "EJEMPLO"

Ready
```

La longitud de una variable de tira de caracteres no está limitada. Estas variables pueden tener subíndices, como pasa con las variables reales y enteras.

Ejemplo:

```
10      DIM E8$ (2,7)    , A$ (13%)

Ready
```

El Basic Plus comprende un cierto número de *funciones intrínsecas* que definen operaciones con las variables de tiras de caracteres. A continuación veremos las más usadas.

Concatenación de varias variables de tira de caracteres.

Ejemplo:

```
10      A$ = "ABC" \ B$ = "DEF"
20      C$ = A$ + B$ \ D$ = B$ + "X  YZ"
30      PRINT A$ \ PRINT B$ \ PRINT C$ , D$

Ready

RUNNH
ABC
DEF
ABCDEF          DEFX   YZ

Ready
```

La función ASCII (A$) da el valor ASCII del primer carácter de la tira

Ejemplo:

```
10      INPUT "TIRA DE CARACTERES"; A7$
20      PRINT "VALOR ASCII ="; ASCII (A7$)

Ready
```

```
RUNNH
TIRA DE CARACTERES? ENSAYO
VALOR ASCII = 69

Ready
```

La función CHR$(N%) genera o entrega un carácter cuyo código ASCII es el valor de N.

Ejemplo:

```
10      INPUT "NO DE CARACTER ASCII"; N%
20      PRINT N%; 'REPRESENTA EL CARACTER " '; CHR$ (N%); ' " '

Ready

RUNNH
NO DE CARACTER ASCII? 65
  65 REPRESENTA EL CARACTER "A"

Ready
```

La siguiente función se usa a menudo para hacer sonar el timbre que existe en el terminal.

```
10      PRINT CHR$ (7%);

Ready
```

La función STRING$ (N1%, N2%) genera una tira de N1 caracteres de longitud, todos ellos iguales al ASCII N2.

Ejemplo:

```
10      INPUT "LONGITUD"; N1%
20      INPUT "NO CARACTER ASCII"; N2%
30      PRINT STRING$ (N1%, N2%)

Ready

RUNNH
LONGITUD? 15
NO CARACTER ASCII? 45
---------------

Ready
```

Se la utiliza en las impresiones controladas, para generar los cuadros o tablas, para subrayar los títulos, etc...

La función LEFT (A$, N%) que selecciona una subtira de la tira de caracteres A$, tomando los N primeros caracteres de la izquierda.

Ejemplo:

```
10      A$ = "ABCDEF  \   B$ = "XYZ"
20      PRINT LEFT (A$, 4%) \ PRINT LEFT (B$, 4%)
\       PRINT LEFT (A$, 3%) + LEFT (B$, 1%)

Ready

RUNNH
ABCD
XYZ
ABCX

Ready
```

La función RIGHT (A$, N%), análoga a la anterior, selecciona una subtira de la tira de caracteres A$, partiendo del carácter Nsimo y llegando hasta el extremo derecho de la misma.

Ejemplo:

```
10      A$ = "ABCDEF"  \  B$ = "XYZ"
20      PRINT RIGHT (A$, 4%)   \ PRINT RIGHT (B$, 4% )
\       PRINT RIGHT (A$, 2%)   + RIGHT (B$, 3%)

Ready

RUNNH
DEF

BCDEFZ

Ready
```

La función MID (A$, N1 % , N2 %) selecciona una subtira de la cadena de caracteres A$, comenzando en su N1-simo caracter y comprendiendo N2 caracteres.

Ejemplo:

```
10      A$ = "ABCDEF"  \  B$ = "XYZ"
20      PRINT MID(A$, 2%, 3%)
\       PRINT MID(A$, 3%, 2%)
\       PRINT MID(B$, 5%, 5%)
\       PRINT MID(A$, 1%, 2%) + MID(B$, 2%, 2%)

Ready

RUNNH
BCD
CD

ABYZ

Ready
```

La función LEN(A$) entrega un valor entero que es la longitud de la tira A$.

Ejemplo:

```
10      INPUT A$
20      PRINT LEN(A$)
30      GOTO 10

Ready

RUNNH
? ENSAYO
  6
? BASIC-PLUS
  10
? ^C

Ready
```

La función INSTR (N1, A$, B$) que busca en la tira A$ a partir de su N1-simo carácter, sí contiene el primer carácter de la tira B, dando su

posición. Si la tira B$ es vacía, la función devuelve 1, pero si el primer carácter de B$ no aparece en A$, la función da 0.

Ejemplo:

```
10      A$ = "ABCDEFGHIJ" \ B$ = "GH" \ C$ = "XY" \ D$ = " "
20      PRINT INSTR(1, A$, B$)
\       PRINT INSTR(8, A$, B$)
\       PRINT INSTR(1, A$, C$)
\       PRINT INSTR(1, A$, D$)

Ready

RUNNH
 7
 0
 0
 1

Ready
```

La función VAL (A$) entrega el valor de una tira de caracteres *numéricos*. Si la tira A$ no es numérica, dará un error, entregando el valor 0.

Ejemplo:

```
10      A$ = "12345"  \  B$ = "1234.567"
20      X% = VAL (A$)  \  X = VAL(B$)
30      PRINT X%  ,  X

Ready

RUNNH
 12345              1234.57

Ready
```

La función NUM$ (N) da la representación de un numero como tira de caracteres (lo contrario de la función anterior). La longitud de la tira resultante será igual al número de cifras del número original (más eventualmente una o dos posiciones adicionales para la coma y el signo si el

número fuera negativo), también se contabilizan dos posiciones o blancos que separan al número de lo que le rodea.

Ejemplo:

```
10      N = 1234.56
20      A$ = NUM$(N)
30      PRINT "*"; A$; "*"

Ready

RUNNH
*  1234.56  *

Ready
```

La función NUM1$(N) es como la anterior pero no da los dos blancos separadores.

Ejemplo:

```
10      N = 1234.56
20      A$ = NUM1$(N)
30      PRINT "*"; A$; "*"

Ready

RUNNH
*1234.56*

Ready
```

3.3 LAS RUPTURAS DE SECUENCIA

El abanico de posibilidades de órdenes de salto condicional es muy rico en Basic Plus. Además de la orden IF . . . THEN ya vista se encuentran las

1ª ON expresión GOTO nº de línea 1, nº de línea 2, ...

2ª ON expresión GOSUB nº de línea 1, nº de línea 2, ...

3ª IF condición THEN instrucción ELSE instrucción(es)

4ª instrucción IF condición

La primera instrucción permite bifurcaciones a líneas diferentes según el valor de una variable o de una expresión. La segunda instrucción en vez de bifurcaciones ejecuta subprogramas. La tercera instrucción permite realizar acciones adecuadas cualquiera que sea el resultado de una comparación. Además permite poner varias instrucciones después del ELSE (inclusive otra instrucción IF. . . THEN. . . ELSE), en cambio después del THEN sólo puede ir una instrucción que no sea IF. La última instrucción es un caso particular del salto condicional. En efecto según el valor de la condición que sigue a IF se ejecuta o no la instrucción que le precede pasando a la instrucción siguiente.

Ejemplo 1:

```
10      INPUT "VALOR DE X"; X
20      ON X GOTO 100,110,120
100     PRINT "LINEA 100  X="; X  \  GOTO 10
110     PRINT "LINEA 110  X="; X  \  GOTO 10
120     PRINT "LINEA 120  X="; X  \  GOTO 10
32767   END

Ready

RUNNH
VALOR DE X? 1
LINEA 100  X= 1
VALOR DE X? 2
LINEA 110  X= 2
VALOR DE X? 3
LINEA 120  X= 3
VALOR DE X? 4
?ON statement out of range at line 20

Ready
```

Ejemplo 2:

```
10      INPUT "VALOR DE X"; X
20      ON X GOSUB 100, 110, 120
30      GOTO 10
```

```
100     PRINT "LINEA 100 X="; X    \ RETURN
110     PRINT "LINEA 110 X="; X    \ RETURN
120     PRINT "LINEA 120 X="; X    \ RETURN
32767   END

Ready

RUNNH
VALOR DE X? 1
LINEA 100  X = 1
VALOR DE X? 2
LINEA 110  X = 2
VALOR DE X? 3
LINEA 120  X = 3
VALOR DE X? 0
?ON statement out of range at line 20
```

Ejemplo 3:

En función del número del día, del mes y del año, se quiere calcular el día de la semana. Sea I el número del día, M el mes, Y el año. El algoritmo utilizado es el siguiente (1).

$$\text{nº del día de la semana} = \text{módulo } 7(I - 1 + \text{INT}(5Y1/4) - \text{INT}(Y1/100) + \text{INT}(Y1/400) + \text{INT}(13(1 + M1)/5))$$

siendo

$$Y1 = Y - 1 \text{ y } M1 = M + 12 \quad \text{si } M = 2$$
$$Y1 = Y \quad \text{y } M1 = M \quad \text{en caso contrario}$$

```
10      DIM J$ (6%)
20      FOR I% = 0% TO 6%
\       READ J$ (I%)
\       NEXT I%
30      DATA DOMINGO, LUNES, MARTES, MIERCOLES, JUEVES, VIERNES,
        SABADO
35      REM  LECTURA Y CONTROL DE LA FECHA
40      INPUT "NO DE DIA"; J%
```

(1) Se considera el comienzo del año el 1º de marzo, por lo que los meses de enero y febrero son los 13º y 14º del año anterior. Con INT se indica la parte entera. Módulo es el resto de la división entera de dos variables. Por ejemplo: Módulo 7(22) = 1.

```
\       GOTO 40 IF J% < 1% OR  J% >  31%
50      INPUT "NO DE MES"; M%
\       GOTO 50 IF M%< 1 % OR  M % > 12%
60      IF (M% = 4% OR M%= 6 %OR M % = 9% OR M%= 11%) AND  J% >
        30 % THEN 40 ELSE IF M % = 2 % AND  J % > 29 %THEN 40
70      INPUT "ANUAL"; Y %
\       GOTO 70 IF Y % < 1 %
80      IF(Y/4 – INT(Y/4) <> 0) AND (M % = 2%AND J % > 28%)
        THEN 40
85      REM  A L G O R I T M O
90      M1% = M%
\       Y1% = Y %
\       IF M % <= 2 % THEN M1% = M1% + 12%
                          \ Y1% x Y1% – 1%
100     J1% = J% – 1 + INT (5*Y1% /4%) – INT (Y1% /100%) + INT (Y1%
        /400 % ) + INT (13* (1 + M1% )/5%)
110     J1% = J1% – INT (J1%/7 %)*7 %  ! CALCULO DEL MODULO 7
120     PRINT "EL"; J%; "/"; M % ; "/" % ES UN "; J$ (J1%)
32767   END

Ready

RUNNH
NO DE DIA? 1
NO DE MES? 1
ANUAL? 1982
EL 1 / 1 / 1982  ES UN VIERNES

Ready
```

3.4 LOS BUCLES DE PROGRAMA

Además de la instrucción clásica

FOR variable = expresión TO expresión STEP expresión

tenemos en Basic Plus otras posibilidades de formación de bucles de programa, en especial las instrucciones:

FOR variable = expresión STEP expresión WHILE condición
FOR variable = expresión STEP expresión UNTIL condición

Además existe una forma especial de bucles de programa que áfectan a una sola instrucción del programa.

Ejemplo 1:

```
.10     FOR X = 1 STEP 2 WHILE Z.< 30
20      Z = X**2
30      PRINT X, Z
40      NEXT X
32767   END

Ready

RUNNH
 1              1
 3              9
 5              25
 7              49

Ready
```

En este ejemplo las líneas 20 y 30 se ejecutarán mientras la condición sea verdadera.

Ejemplo 2:

```
10      FOR X = 1 STEP 2 UNTIL Z >= 30
20      Z = X**2
30      PRINT X, Z
40      NEXT X
32767   END

Ready
```

este ejemplo no se diferencia del anterior más que en la línea 10; las líneas 20 y 30 se ejecutarán hasta que la condición sea verdadera.

Las notaciones WHILE y UNTIL se emplean cuando la salida del bucle depende de una variable diferente del índice del bucle.

Los bucles que sólo afectan a una instrucción tienen una de las cuatro formas siguientes:

instrucción FOR variable = expresión TO expresión STEP expresión

instrucción FOR variable = expresión STEP expresión $\begin{matrix}\text{UNTIL}\\\text{WHILE}\end{matrix}$ condición.

instrucción UNTIL condición

instrucción WHILE condición

3.5 LOS OPERADORES LOGICOS

Los operadores lógicos se utilizan en ciertas operaciones lógicas sobre variables enteras, pero sobre todo en las instrucción IF-THEN.

En el Basic Plus existen 6 operadores lógicos:

NOT	negación lógica
AND	producto lógico
OR	suma lógica
XOR	disyunción lógica
IMP	implicación
EQV	equivalencia lógica

en la práctica los operadores más usados son AND y OR.

Las dos tablas siguientes resumen las operaciones posibles con estos operadores (1).

A	NOT A
V	F
F	V

A	B	A AND B	A OR B	A XOR B	A IMP B	A AQV B
V	V	V	V	F	V	V
V	F	F	V	V	F	F
F	V	F	V	V	V	F
F	F	F	F	F	V	V

(1) Las expresiones lógicas no pueden tomar más que dos valores, verdadero (V) o falso (F). En las operaciones con variables enteras, conviene sustituir V por 1 y F por 0.

Ejemplo:

```
10      INPUT "INTRODUZCA 3 NUMEROS EN ORDEN CRECIENTE"; A, B, C
20      IF A < B AND B < C THEN PRINT "GRACIAS"
        ELSE PRINT "REPITA" \ GOTO 10
32767   END

Ready

RUNNH
INTRODUZCA 3 NUMEROS EN ORDEN CRECIENTE  1, 3, 2
REPITA
INTRODUZCA 3 NUMEROS EN ORDEN CRECIENTE  5, 4, 3
REPITA
INTRODUZCA 3 NUMEROS EN ORDEN CRECIENTE  1, 2, 3
GRACIAS

Ready
```

Cuando se usen operadores lógicos sobre variables enteras, las operaciones lógicas se efectúan sobre los bits de las palabras correspondientes.

Ejemplo:

El siguiente programa es muy útil cuando es necesario averiguar si un número dado es par o impar.

```
10      INPUT "INTRODUZCA UN ENTERO"; I%
20      IF (I% AND 1%) = 1% THEN PRINT I%; "ES IMPAR"
        ELSE PRINT I%; "ES PAR"
32767   FIN

Ready

RUNNH
INTRODUZCA UN ENTERO? 123
 123 ES IMPAR

Ready

RUNNH
```

```
INTRODUZCA UN ENTERO? 4
 4 ES PAR

Ready
```

3.6 SUBPROGRAMA DE CORRECCION (PROGRAMABLE) DE ERRORES

Durante la ejecución de un programa, el sistema puede encontrarse con errores (p. ej. división por cero, entrada en un INPUT de un valor incorrecto, etc.).

Normalmente, en caso de error, el sistema imprime un mensaje de error y para la ejecución del programa. El Basic Plus ofrece la posibilidad de corrección de ciertos errores sin detener la ejecución del programa, empleado en subprograma especial. Para ello, todos los errores van numerados y se dispone de dos variables reservadas ERR y ERL que contienen, cuando se produce un error, el número del error y la línea en que se produjo.

Para poder usar el subprograma de corrección de errores, es necesario indicárselo al sistema con la instrucción:

ON ERROR GOTO número de línea

que se sitúa antes de toda instrucción que pueda llamar a este subprograma. El subprograma de error comienza en la línea que se especifica en la instrucción ON.

En este subprograma se emplea a menudo la instrucción:

RESUME número de línea

que permite salir del programa de error y seguir la ejecución del programa en la línea especificada en esta instrucción.

Ejemplo:

```
5       ON ERROR GOTO 32700
10      PRINT "INTRODUZCA UN ENTERO"
20      INPUT J
30      PRINT "GRACIAS"  \ GOTO 32767
32700   IF ERR = 50 AND ERL = 20 THEN PRINT "UN ENTERO, POR FAVOR"
```

```
\       RESUME
32767   END

Ready

RUNNH
INTRODUZCA UN ENTERO? ABCD
UN ENTERO, POR FAVOR
? 1.234
UN ENTERO, POR FAVOR
? 5
GRACIAS

Ready
```

3.7 COMPLEMENTOS SOBRE INSTRUCCIONES UTILES

Veremos aquí algunas instrucciones diversas, pero que tienen como característica común su uso frecuente.

INPUT LINE variable de tira de caracteres

esta instrucción es similar a la instrucción INPUT y permite la entrada de una secuencia de caracteres a partir de un terminal (1). El final de la tira se señala con RETURN (los caracteres ASCII son 13 y 10), estos dos caracteres forman parte también de la tira.

Ejemplo:

```
10      PRINT "INTRODUZCA UN TEXTO"
20      INPUT LINE A$
30      PRINT A$ \ PRINT "LONGITUD DEL TEXTO ="; LEN (A$)
32767   END

Ready

RUNNH
INTRODUZCA UN TEXTO? ENSAYO
```

(1) Y también de un fichero (ver el capítulo 4, Los ficheros)

```
ENSAYO

LONGITUD DEL TEXTO = 8

Ready
```

El Basic Plus ofrece la posibilidad de definir funciones sobre varias líneas, de la forma siguiente:

```
DEF FNidentificador lista de argumentos
definición de la función
FNEND
```

Las funciones y sus argumentos pueden ser de tipos cualesquiera:

Ejemplo:

```
10      DEF FNF (R%)
20      IF R%= 0% THEN  FNF  = 1% \ GOTO 40
30      IF R%= 1% THEN FNF  = 1%ELSE FNF = R% * (R% – 1%)
40      FNEND
100     INPUT "INTRODUZCA UN ENTERO"; R
110     PRINT "EL FACTOR DE"; R% ; "ES IGUAL A"; FNF (R%)
32767   END

Ready

RUNNH
INTRODUZCA UN ENTERO? 6
EL FACTOR DE  6 ES IGUAL A 30

Ready
```

Ejemplo:

```
10      DEF FNC   (A$, B$)
20      IF A$ <= B$ THEN C$ = A$ + "XXX" + B$ ELSE C$ = A$ +
        "YYY" + B$
30      FNC% = LEN (C$)
40      FNEND
```

```
100     INPUT "X$ ="; X$
110     INPUT "Y$ ="; Y$
120     PRINT FNC   (X$, Y$) , C$
32767   END

Ready

RUNNH
X$ = ? A
Y$ = ? B
 5                AXXXB

Ready
```

Cuando los programas crecen o se hacen más importantes, o bien hay partes que son idénticas en varios programas, es posible bifurcar a otro programa, mediante la instrucción CHAIN nombre del programa, número de línea.

Ejemplo:

```
17000   CHAIN "ORIGEN" 31000

Ready
```

Al llegar a la línea 17000, el programa le pasará la vez a la línea 31000 del programa ORIGEN.

CAPITULO 4

Los ficheros

4.1 NOCION DE FICHERO

El concepto de fichero es un concepto fundamental que aparece en todas las aplicaciones de gestión.

Un fichero es una colección organizada de informaciones que tienen entre ellos un enlace lógico y que pueden ser consultadas individualmente de manera iterativa y sistemática. Por consiguiente, un fichero no es solamente un conjunto de datos usados por un programa, sino que también lo es el propio programa.

Un fichero, formado por una serie de registros, se almacena en un soporte material (disco magnético, cinta magnética, cinta o ficha perforada, etc.). Al hablar de registro, hay que distinguir entre registro lógico y registro físico.

El *registro lógico* es el conjunto de informaciones relativo a las entidades elementales que son objeto de un tratamiento individual en el proceso iterativo al que se somete el fichero. En cambio *registro físico* es la unidad de grabación o registro en el almacenamiento.

Un fichero se identifica con un cierto número de etiquetas, por ejemplo en Basic Plus (1) todo fichero se identifica con un nombre (2) (de 6 o más caracteres), la extensión y algunas veces se incluye la identificación del periférico en el que está situado. La extensión está formada por un punto seguido de 1 a 3 caracteres alfanuméricos; y la crea el

(1) Empleado con el sistema operativo RSTS.

(2) En realidad, sólo los ficheros almacenados sobre periféricos con ficheros estructurados (disco y cinta magnética) llevan un nombre. Los ficheros grabados en periféricos con ficheros no estructurados (impresora, lectora/perforadora de cinta, lectora/perforadora de fichas) se direccionan especificando el periférico.

sistema (por ejemplo, los programas fuente de Basic Plus tienen siempre la extensión BAS, y una vez compilados su extensión es BAC), o bien el usuario puede escoger la extensión.

En Basic Plus hay tres tipos de ficheros:

- en la forma ASCII
- en entradas/salidas por registro
- en memoria virtual.

4.2 APERTURA Y CIERRE DE UN FICHERO

Antes de realizar cualquier operación con un fichero es necesario abrirlo. La apertura de un fichero realiza la asignación a este fichero de un número de canal de entrada/salida (de 1 a 12). Durante la ejecución del programa, cada referencia a un fichero de datos, se hace a través de este número.

La instrucción de apertura de un fichero se escribe de esta forma:

OPEN Nombre de fichero [FOR INPUT / FOR OUTPUT] AS FILE Designación de periférico o canal [RECORDSIZE expresión]

```
100     OPEN "TOTO.DAT" AS FILE 2%
110     X% = 3%
\       OPEN "STOCK. FIC" AS FILE X%
120     A$ = "DRO: COMPTA. LST"
\       OPEN A$ FOR OUTPUT AS FILE 4% ! Fichero situado en el disco DRO
130     OPEN "LP:" AS FILE 5          ! Apertura de la impresora

Ready
```

Las opciones FOR INPUT y la FOR OUTPUT permiten abrir un fichero ya existente o uno nuevo, respectivamente.

La opción FOR INPUT hace que el sistema busque el fichero especificado. Si el fichero se encuentra se abre haciéndolo disponible, pero si no el sistema da un error de "fichero no existente" (CAN'T FIND FILE OR ACCOUNT).

Con la opción FOR OUTPUT se abre el fichero especificado, existiera antes o no. Si el fichero ya existe, su contenido se destruye antes de

la apertura. La omisión de la opción hace que se abra el fichero en todo caso, sin alteración de su contenido.

Todos los intercambios de información entre el fichero y el programa se hacen mediante una zona intermedia, reservada por el sistema en la memoria central para cada canal de entrada/salida. Esta zona intermedia se llama zona de paso o *buffer*, y corresponde en realidad al tamaño de un registro físico.

La opción RECORDSIZE permite fijar el tamaño del buffer. Sin embargo al omitir la opción RECORDSIZE, el sistema asigna valores por defecto a los buffers, adecuados a cada periférico. El tamaño del buffer asociado por el sistema a un disco o armario de cinta magnética es de 512 caracteres. Esta opción se utiliza normalmente cuando el tamaño del registro lógico es superior al tamaño del registro físico.

Una vez se han terminado los intercambios de información entre el programa y el fichero, se debe cerrar el fichero. El cierre destruye los enlaces lógicos entre el fichero y su canal asociado. Cuando un fichero se ha empleado en salida, el cierre tiene por efecto además la escritura de la zona de comunicación en el fichero.

La forma general de esta instrucción es la siguiente:

CLOSE expresión , expresión .....

en la que expresión es un entero comprendido entre 1 y 12.

Ejemplo:

```
1000  CLOSE 1% , 7% ! Cierre canal 1 y 7
1100  CLOSE I% FOR I% = 1 % TO 12% ! Cierre de todos los canales

Ready
```

4.3 FICHEROS EN FORMA ASCII

Es la forma más sencilla de organización de ficheros. Sólo son posibles la lectura y escritura en forma secuencial (1).

(1) La escritura se puede hacer en todos los soportes (todos los soportes magnéticos, las perforadoras de cinta, impresora, consola de rayos catódicos). La lectura en cambio, sólo es posible en ciertos soportes (soportes magnéticos, lectoras de fichas, cinta, teclado). En lo sucesivo, supondremos que el fichero está situado en un soporte de acceso directo (disco).

La escritura de ficheros ASCII se realiza con ayuda de la instrucción:

PRINT # expresión, lista

la expresión representa el número de canal de entrada/salida. La lista contiene las variables, constantes y las expresiones se separan con puntos y comas (;) o bien comas (,) definiendo así el formato de escritura. Dichas variables se representan siempre en forma de caracteres, por lo que los números reales y enteros se tienen que convertir a tiras de caracteres, función que realiza automáticamente el sistema.

En los ficheros ASCII los registros se separan entre sí por caracteres RETURN (caracteres ASCII 13 y 10). Un registro se puede escribir con una o varias instrucciones PRINT, y el RETURN de separación lo genera la instrucción PRINT, en la que el último elemento de la lista no va seguido de un signo de puntuación (; o bien ,).

Ejemplo:

```
110     PRINT A$;
120     PRINT B$            ¡Escritura de un registro con dos PRINTs
130     PRINT A$; C$        ¡Escritura de un solo registro
140     PRINT B$            ¡Escritura de un registro

Ready
```

La lectura de ficheros bajo forma ASCII se efectúa mediante la instrucción

INPUT LINE # (expresión), (variable de tira de caracteres)

en donde expresión representa, como en la instrucción anterior, el número de canal. Esta instrucción lee el fichero hasta que encuentra un RETURN. El resultado de esta lectura se coloca en la variable de tira de caracteres.

Así varias variables o constantes escritas con una sola o con varias instrucciones PRINT pueden leerse con una sola instrucción INPUT LINE.

Ejemplo:

```
90      ! Escritura de un fichero ASCII
        !
100     OPEN "DBO: ENSAYO. DAT" AS FILE 1%
        ! Apertura de un fichero en disco DBO
```

```
110     A$ = "ENSAYO"
120     FOR I% = 1 % TO 3 %
130     PRINT # 1 %, "NO"; I %; 100 % + I %; 1000 % + I %
140     PRINT # 1, A$
150     NEXT I %
160     CLOSE 1 %   ! Cierre del fichero ⇒ escritura del último bloque
        !
        ! LECTURA DEL FICHERO
        !
170     OPEN "DBO: ENSAYO. DAT" AS FILE 1 %    ! Reapertura del fichero
180     FOR I % = 1 % TO 6 %
190     INPUT LINE # 1, A$
200     PRINT A$
210     NEXT I
32767   END

Ready
```

En este ejemplo, después de la apertura del fichero ENSAYO.DAT del disco DBO se escriben, en cada bucle del programa, dos registros: el primero lleva 4 elementos (la constante NO y 3 variables: I % , I % + + 100 % y I %+ 1000 %)en línea 130; el segundo sólo lleva un elemento en la línea 140. Luego, después de cerrar el fichero para escribir el último buffer, se le abre de nuevo y se procede a la lectura e impresión de los registros escritos anteriormente.

Ejemplo:

```
RUNNH
NO   1    101      1001

ENSAYO

NO   2    102      1002

ENSAYO

NO   3    103      1003

ENSAYO

Ready
```

La ventaja principal de los ficheros en forma ASCII es su facilidad de empleo y el inconveniente mayor es la organización secuencial.

4.4 FICHEROS CON ENTRADAS/SALIDAS POR REGISTRO

Los ficheros con entradas/salidas por registro constituyen la forma más flexible y también la más compleja de uso de ficheros en Basic Plus.

El acceso a los ficheros E/S por registro puede hacerse en orden secuencial o bien directamente mediante el orden de grabación. Se pueden mezclar dentro de un registro variables de todos los tipos.

4.4.1 Lectura y escritura de un fichero E/S por registro: instrucciones GET y PUT

Las instrucciones GET y PUT permiten efectuar intercambios de información entre el fichero y la zona intermedia que tiene asociada.

La instrucción GET cuyo papel es leer y transferir información desde el fichero al buffer, tiene el formato siguiente:

GET # (expresión 1) [, RECORD expresión 2]

la expresión 1 indica el número de canal y la expresión 2 el número de registro físico que se quiere leer.

Cuando no está la opción RECORD, la lectura del fichero se hace en orden secuencial, es decir que con cada GET, el sistema leerá el registro siguiente y lo pone en el buffer. Pero si esta opción está presente, permite el acceso directo a cualquier registro del fichero.

En efecto, los bloques físicos del fichero (cada uno tiene 512 caracteres de longitud) van numerados de 1 a N y con la expresión de la opción RECORD se indica el número del bloque que debe leerse. El tamaño del registro leído depende de la opción RECORDSIZE precisada en la instrucción OPEN, y que no puede ser inferior al tamaño mínimo del buffer (512 caracteres).

Ejemplo:

```
100     GET#3, RECORD I % FOR I % = 1 % TO 100 %  STEP 5%
110     GET#4

Ready
```

La escritura del contenido del buffer al fichero se hace con la ayuda de la instrucción PUT

PUT # expresión 1 [, RECORD expresión 2]

y el sentido de las expresiones 1 y 2 es idéntico al de la instrucción anterior.

El tamaño de la zona intermedia depende, como en la instrucción GET, de la opción RECORDSIZE.

En el caso de que la longitud del registro lógico sea inferior al tamaño de la zona de transferencia, una instrucción GET (o PUT) puede transferir hacia (desde) el fichero, varios registros lógicos cuando estos están agrupados.

La utilización de la opción RECORD es idéntica a la vista en GET.

Ejemplo:

```
100     PUT#3%
110     PUT#N%,RECORD 254%

Ready
```

4.4.2 Acceso a la zona de transferencia de los ficheros de E/S por registros

La posibilidad de transferencia de información desde (hacia) los ficheros no es suficiente. Es necesario además tener acceso a la información con el fin de consultarla y modificarla. Estas funciones se realizan con las instrucciones

FIELD, LSET y RSET

la forma general de la primera instrucción es

FIELD # (expresión), expresión 1 AS vtc (1), expresión 2 AS vtc 2, ...

en la que expresión define el número de canal y expresión *n* define la longitud de la variable de tira de caracteres n.

(1) Variable de tira de caracteres.

Esta instrucción permite crear enlaces lógicos entre los nombres de las variables de tira de caracteres y la totalidad o partes de la zona de transferencia asociada a un fichero. La instrucción FIELD no tiene ninguna acción física sobre el registro: no es más que colocar una plantilla en la zona intermedia.

Consideremos un registro de longitud de 128 caracteres compuesto de la forma siguiente:

```
100     OPEN "FICH. DAT" AS FILE 10%
110     FIELD #10%, 30 % AS N$ , 20  %  AS  P$   , 78 % AS A$

 Ready
```

En el resto del programa, N$ corresponderá a los primeros 30 caracteres del registro, P$ a los siguiente 20 caracteres y A$ a los 78 caracteres que van después de P$. Con la instrucción FIELD empleada en el ejemplo, no se definen más que los primeros 128 caracteres de la zona intermedia, que aquí se supone igual a 512 caracteres, dejando sin emplear los 384 caracteres siguientes.

Con el fin de usar mejor todos los caracteres del registro físico, basta agrupar varios registros lógicos en un registro físico.

```
100  OPEN "FICH. DAT" AS FILE 10
110  FIELD #10%,  30%  AS N$ (0 % ), 20 %  AS P$ (0 % ), 78 %  AS A$ (0 % )
                , 30 % AS N$ (1 % ), 20 % AS P$ (1 % ), 78 % AS A$ (1 % )
                , 30 % AS N$ (2 % ), 20 % AS P$ (2 % ), 78 % AS A$ (2 % )
                , 30 % AS N$ (3 % ), 20 % AS P$ (3 % ), 78 % AS A$ (3 % )

Ready
```

La instrucción FIELD de la línea 110 se escribe también así:

```
100     FIELD #10%, I% *  128 %  AS Z$,
        30      AS N$ (I % ),
        20      AS P$ (I % ),
        78      AS A$ (I % )      FOR I %  = 0 %TO 3 %

 Ready
```

Una vez definidas las variables de tira de caracteres en la zona intermedia, es posible asignarle valores. Esta operación debe hacerse con ayuda de las instrucciones:

LSET < variable de tira de caracteres > = < Tira >
RSET < variable de tira de caracteres > = < Tira >

Estas dos instrucciones posicionan, en la variable definida sobre la zona intermedia, una constante o variable de tira de caracteres, borrando primeramente el contenido anterior de la variable. Las instrucciones LSET y RSET no modifican la longitud de la variable definida anteriormente.

Así, si la nueva tira es de mayor longitud se truncará, si por el contrario es más corta, se completará con blancos:

- por la derecha con la instrucción LSET
- por la izquierda con la instrucción R SET.

Dicho de otra forma, la nueva tira de caracteres será colocada a la izquierda con LSET y a la derecha con RSET.

4.4.3 Ejemplo de utilización de un fichero con entradas/salidas por registros

Enunciado del problema:

Se trata de manejar un fichero de personal cuyos registro se definen de la forma siguiente:

- apellidos 30 caracteres
- nombre 20 caracteres
- dirección 78 caracteres

El acceso al fichero se hace por un campo o matrícula numérica cuyo valor puede variar de 1 a 1000.

Las operaciones a efectuar con el fichero son las siguientes:

- creación de un registro completo
- modificación del registro, zona a zona
- supresión del registro.

El diseño del registro físico y lógico se ha dado en el ejemplo anterior.

```
5       ON ERROR GOTO 32700
10      OPEN "PERSON. DAT" AS FILE 1%
20      FIELD #1%, I% * 128% AS Z$, 30% AS N$(I%), 20% AS P$
        (I%), 78% AS A$(I%)
        FOR I% = 0% TO 3%
30      PRINT "(C) reación, (M)odificación, (S)upresión, (F)in trabajo"
40      INPUT "Elija Vd"; C$
\       GOTO 32760 IF C$ = "F"
\       GOTO 40 IF C$ <> "C" AND C$ <> "M" AND C$ <> "S"
100     INPUT "MATRICULA"; M%
\       GOTO 100 IF M% < 1% OR M% > 1000%
110     M1% = (M% - 1%)/4% + 1%
\       M2% = M% - (M1% - 1%)*4% - 1%
120     GET #1, RECORD M1%
130     GOTO 200 IF C$ = "M" OR C$ = "S"
        !
        ! C R E A C I O N
        !
140     IF N$(M2%) <> SPACE$(30%) THEN PRINT "MATRICULA YA
        EXISTE" \GOTO 30
150     INPUT "APELLIDO";N1$
\       GOTO 150 IF N1$ = ""
\       INPUT "NOMBRE"; P1$
\       PRINT "DIRECCION";
\       INPUT LINE A1$
\       A1$ = LEFT(A1$, LEN(A1$) - 2%)
160     LSET N$(M2%) = N1$
\       LSET P$(M2%) = P1$
\       LSET A$(M2%) = A1$
170     PUT #1%, RECORD M1%
\       GOTO 30
        !
        !
200     IF N$(M2%) = SPACE$(30%) THEN PRINT "MATRICULA NO
        EXISTE" \GOTO 30%
210     PRINT
\       PRINT "1. APELLIDO  : "; N$(M2%)
\       PRINT "2. NOMBRE    : "; P$(M2%);
\       PRINT "3. DIRECCION : "; A$(M2%);
\       PRINT
```

```
\       GOTO 300 IF C$ = "S"
220     !
        ! M O D I F I C A C I O N
        !
230     INPUT "NO DE CAMPO A MODIFICAR"; I%
\       GOTO 280 IF I % <= 0 %
\       GOTO 230 IF I % > 3 %
240     ON I % GOTO 250, 260, 270
250     INPUT "NUEVO APELLIDO"; N1$
\       GOTO 230 IF N1$ = "    "
\       LSET N$ (M2 % ) = N1$
\       GOTO 230
260     INPUT "NUEVO NOMBRE"; P1$
\       LSET P$ (M2 % ) = N1$
\       GOTO 230
270     PRINT "NUEVA DIRECCION"; P1$
\       INPUT LINE A1$
\       A1$ = LEFT (A1$, LEN (A1$) – 2 % )
        LSET A$(M2%) = A1$
\       GOTO 230
280     GOTO 320
300     !
        ! S U P R E S I O N
        !
310     INPUT "SUPRESION (SI/NO)"; C$
\       GOTO 30 IF C$ <> "SI"
\       LSET N$ (M2 % ) = " "
\       LSET P$ (M2 % ) = " "
\       LSET A$ (M2 % ) = " "
320     PUT #1, RECORD M1%
\       GOTO 30
32700   IF ERR = 50 OR ERR = 52 THEN RESUME
32710   IF ERR = 11 AND C$ = "C" THEN RESUME 150
        ELSE PRINT "MATRICULA NO EXISTE"    \ RESUME 30
32760   CLOSE 1%
32767   END

Ready

RUNNH
(C)reación, (M)odificación, (S)upresión, (F)in trabajo
```

```
Elija Vd? C
MATRICULA? 100
APELLIDO? CERVANTES
NOMBRE? JUAN
DIRECCION? P. CASTELLANA, 4 MADRID-1
(C)reación, (M)odificación, (S)upresión, (F)in trabajo
Elija Vd? C
MATRICULA? 100
MATRICULA YA EXISTE
(C)reación, (M)odificación, (S)upresión, (F)in trabajo
Elija Vd? M
MATRICULA? 100

1. APELLIDO  : CERVANTES
2. NOMBRE    : JUAN
3. DIRECCION: P. CASTELLANA, 4 MADRID-1

NO DE CAMPO A MODIFICAR? 2
NUEVO NOMBRE? JUAN-PEDRO
NO DE CAMPO A MODIFICAR?
(C)reación, (M)odificación, (S)upresión, (F)in trabajo
Elija Vd? F

Ready
```

Después de la apertura del fichero (línea 10) y de la definición de la descomposición de la zona de transferencia (línea 20), se le pide al usuario el elegir el tipo de operación a efectuar y el número de matrícula.

Luego (línea 110) en función del número de matrícula se calcula el número de registro en el que se sitúa el artículo en cuestión (M1 %), así como la posición relativa de este artículo en el interior del registro.

Después de haber leído el fichero (línea 120) se mira el tipo de operación a realizar: si es una creación, se comprueba la disponibilidad del registro con la matrícula dada; para esto, basta mirar si existe el campo del apellido (el apellido es el único campo obligatorio). Si la comprobación es positiva se toman las informaciones, con instrucciones INPUT para nombre y apellidos y con la instrucción INPUT LINE para la dirección. Como la variable A1$ contiene el RETURN, se suprime en la línea siguiente. Después de colocar los datos en el buffer (línea 160), se escribe en el fichero.

Los tratamientos efectuados en las partes MODIFICACION y SUPRESION son casi idénticos a la parte de CREACION.

4.5 LOS FICHEROS EN MEMORIA VIRTUAL

Los ficheros en memoria virtual permiten manejar datos que tengan un volumen importante de una manera muy simple y eficaz. Los datos se disponen en estos ficheros en forma de matriz.

Antes de usar un fichero en memoria virtual hay que abrirlo, como cualquier otro fichero, con la instrucción OPEN.

El hecho de que el fichero se usa como memoria virtual se le indica al sistema con la instrucción:

DIM # (expresión), lista

la expresión indica el número del canal y la lista de variables es idéntica a la lista de una instrucción DIM clásica.

Por ejemplo para usar una matriz de enteros dimensionada de 1000 x x 100 se escribiría:

DIM # 10, Z$ (1000 %, 100%)

También es posible usar ficheros en memoria virtual que contengan tiras de caracteres. Así como en la memoria central las tiras de caracteres pueden tener una longitud cualquiera, los ficheros en memoria virtual exigen la definición de una longitud fija, que además, debe ser una potencia de 2 inferior o igual a 512 (o sea 2, 4, 8, 16, 32, 64, 128, 256 ó 512). La definición de la longitud se hace en la instrucción DIM:

DIM # (expresión), tira de caracteres. (dimensión(es)) = longitud ...

la longitud por defecto, asignada por el sistema en caso de ausencia de precisiones, es de 16 caracteres.

```
15      DIM #10, Z$ (200 % ) = 64 % , G$ (100 % ) = 2 %, D$ (1000 % )

Ready
```

En el ejemplo anterior: la matriz Z$ tiene 201 tiras de caracteres (de 0 a 200), cuya longitud máxima es igual a 64 caracteres; la matriz G$ de 101 elementos de longitud máxima 2 caracteres y la matriz D$ que contiene 1001 elementos con una longitud de 16 caracteres.

Una vez definidas con la instrucción DIM, las tablas en memoria virtual se utilizan como si estuvieran en memoria central.

Comc en todo fichero, se debe cerrar al final de su uso en escritura.

Ejemplo de programa:

Permite la creación y consulta de una tabla que contiene los nombres de 10.000 piezas.

```
10      ON ERROR GOTO 32700
20      OPEN "PIEZAS. VAY" AS FILE 5%
30      DIM#5, N$ (10000 %) = 32 %
40      INPUT "NO PIEZA"; N%
\       GOTO 32700 IF N% <= 0 %
\       GOTO 30 IF N  %  > 10000 %
50      IF N$ (N % ) <> " " THEN PRINT N$  (N%)
\       PRINT
\       GOTO 40
60      PRINT "PIEZA INEXISTENTE"
\       INPUT "TECLEE EL NOMBRE"; N$ (N % )
\       PRINT
\       GOTO 40
32700   IR ERR = 50 OR ERR = 52 THE RESUME
32760   CLOSE 5 %
32767   END
Ready

RUNNH
NO PIEZA? 135
PIEZA INEXISTENTE
TECLEE EL NOMBRE? PIEZA NO. 135

NO PIEZA? 135
PIEZA NO.135

NO PIEZA? 100
PIEZA 100

NO PIEZA?

Ready
```

CAPITULO 5

Problemas de aplicación

Todos los problemas tienen la solución o por lo menos un comienzo de análisis, así como indicación de los párrafos implicados.

5.1 UN METODO DE CLASIFICACION

Enunciado

Ordenar N números en orden creciente.

Método. Comprende los pasos siguientes:

1. Se disponen los N números en un vector I(J).

2. Se tienen dispuestos dos vectores adicionales P(K) y Q(L).

3. Se recorre el vector I(J) y mientras sus elementos están en orden creciente se escriben los elementos de I(J) en P(K), hasta agotar I. Cuando un elemento I(J) no está en orden creciente, se escribe en Q(L) y los elementos sucesivos de I(J) mientras estén en orden creciente se escriben ahora en Q(L); en caso contrario se va a P(K) ... hasta agotar I.

4. Si el vector Q queda vacío, ya se tiene I ordenado. Se imprime el vector P.

5. Si no, se copian los elementos de P y Q en I, comenzando por P(1) y Q(1). A cada etapa se compara P(K) con Q(L) y se escribe en I el menor de ellos. Cuando se acaba uno de los dos P o Q, se escribe el resto en I.

6. Ir al paso 1, hasta que Q quede vacío.

Por ejemplo: I = (45, 67, 1024, 473, 1028, 3, 18, 48, 52)

Etapa 3: P = 45, 67, 1024, 3, 18, 48
Q = 473, 1028, 19, 52

Etapa 4: Q no está vacío

Etapa 5: I_1 = (45, 67, 473, 1024, 1028, 3, 18, 19, 48, 52)

Etapa 3: P_1 = (45, 67, 473, 1024, 1028)
Q_1 = (3, 18, 19, 48, 52)

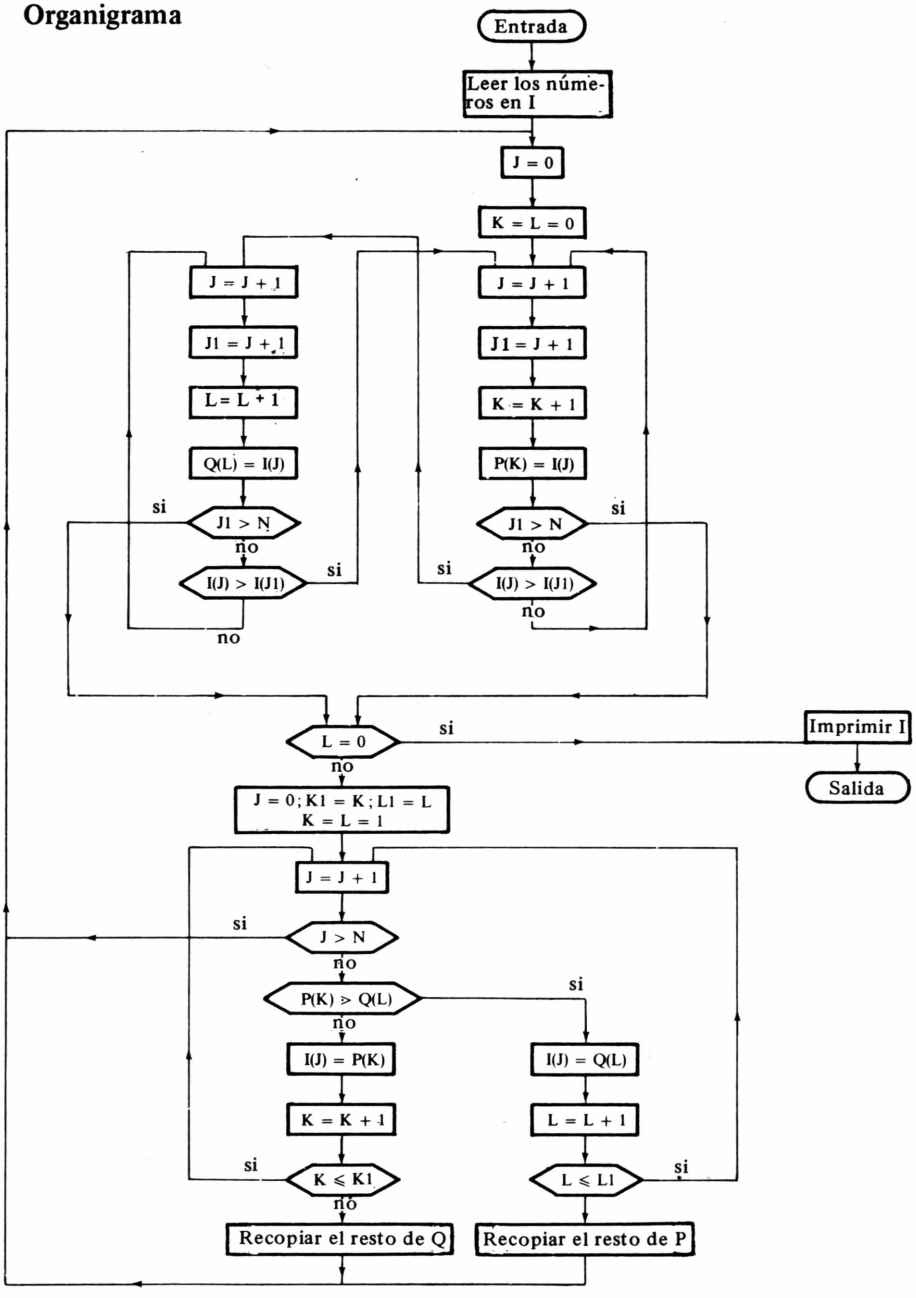
Etapa 4: Q_1 no está vacío

Etapa 5: I_2 = (3, 18, 19, 45, 48, 52, 67, 473, 1024, 1028)
P_2 = (3, 18, 19, 45, 48, 52, 67, 473, 1024, 1028)
Q_2 está vacío

Nota. Este algoritmo no es interesante más que cuando los datos de entrada están casi ordenados, como por ejemplo un fichero en el que se hubieran introducido al azar algunas fichas nuevas.

Consultar el organigrama en la página 88 y el programa en las páginas 89 y 90.

Organigrama



Programa

```
05 PRINT "CLASIFICACION"
10 DIM I(40), P(40), Q(40)
20 READ N
30 FOR J = 1 TO N
40 READ I(J)
50 NEXT J
60 J = 0
61 K = 0
62 L = 0
70 J = J + 1 \ K = K + 1 \ J1 = J + 1
80 P(K) = I(J)
90 IF J1 - N > 0 THEN 270
100 IF I(J) - I(J1) > 0 THEN 180
150 GO TO 70
180 J = J + 1
190 L = L + 1
200 J1 = J + 1
210 Q(L) = I(J)
220 IF J1 - N > 0 THEN 270
230 IF I(J) - I(J1) > 0 THEN 70
240 GO TO 180
270 IF L = 0 THEN 560
290 J = 0 \ K1 = K \ L1 = L
300 K = 1 \ L = 1
330 J = J + 1
340 IF J > N THEN 60
350 IF P(K) > Q(L) THEN 460
370 I(J) = P(K)
380 K = K + 1
390 IF K < = K1 THEN 330
400 FOR L2 = L TO L1
410 J1 = K1 + L2
420 I(J1) = Q(L2)
430 NEXT L2
440 GO TO 60
460 I(J) = Q(L)
470 L = L + 1
480 IF L < = L1 THEN 330
490 FOR K2 = K TO K1
500 J1 = L1 + K2
```

```
510  I(J1) = P(K2)
520  NEXT K2
530  GO TO 60
560  FOR K = 1 TO N
570  PRINT P(K) ;
580  NEXT K
590  STOP
600  DATA 10
610  DATA 45, 67, 1024, 473, 1028, 3, 18, 48, 19, 52
620  END
```

5.2 CLASIFICACION DE DATOS ESTADISTICOS

Enunciado

Se suponen almacenados una serie de datos numéricos no ordenados. Se desea hallar: el número de datos de entrada, la tabla de frecuencias absolutas, relativas y acumuladas para las clases determinadas.

Datos

Para cada serie, se debe tener:

- el número de clases
- el límite superior de la 1ª clase
- los diferentes intervalos
- los datos.

Se emplea el número 99999 para señalar el fin de una serie.

Programa

```
C2      .BAS
 10  PRINT "ANALISIS DE DATOS ESTADISTICOS"
 20  PRINT: PRINT
 30  S1=0
 40  S2=0
 50  D1=99999
 60  D2=-99999
 70  FOR N=1 TO 100
 80  INPUT D
 90  IF D=99999 THEN 170
100  S1=S1+D
110  S2=S2+D↑2
120  IF D > D1 THEN 140
130  D1=D
140  IF D < D2 THEN 160
150  D2=D
160  NEXT N
170  N=N-1
200  M=(D1 - D2)/2
205  IF N = 0  GOTO  240
210  M1=S1/N
220  V = (N * S2 - S1↑2)/N
230  E = SQR(V)
240  PRINT "NUMERO DE ELEMENTOS DE LA SERIE"; N
250  PRINT "VALOR MAXIMO"; D2
260  PRINT "VALOR MINIMO"; D1
270  PRINT "INTERVALO DE VARIACION"; M
280  PRINT "MEDIA"; M1
290  PRINT "DESVIACION-TIPO"; E
300  INPUT R
310  IF R=0 THEN 10
400  END
```

5.3 TRAZADO DE UN HISTOGRAMA

Enunciado

Se tiene un grupo de 1000 personas cuyas estaturas se dan en las 7 clases siguientes

160-165	50
165-170	300
170-175	430
175-180	110
180-185	30
185-190	0
190-195	80

Representar el histograma de esta población.

Método

Debido a la forma de funcionar la impresora (o la pantalla), es más fácil programar las clases verticalmente y las frecuencias horizontalmente.

Programa

```
10  PRINT "HISTOGRAMA"
20  DIM N(7)
25  M1 = 0
30  K = 0
40  FOR I = 1 TO 7
50  READ N(I)
55  IF M1 > N(I) THEN 60
57  M1 = N(I)
60  NEXT I
70  FOR J = 160 TO 190 STEP 5
80  K = K + 1
90  PRINT « DE »; J; « A »; J + 5;
100 PRINT TAB(15); « I »;
120 FOR L = 1 TO N(K)/M1 * 50
130 PRINT « »;
140 NEXT L
150 PRINT N(K)
160 PRINTTAB(15) « I »
170 NEXT J
200 DATA 50, 300, 430, 110, 30, 0, 80
210 END
```

5.4 RESOLUCION DE UN SISTEMA DE ECUACIONES LINEALES

Enunciado

Determinar si un sistema de ecuaciones lineales tiene solución y hallarla.

Método

Sea el sistema:

$$A(1, 1)\,X(1) + A(1, 2)\,X(2) + \cdots + A(1, N)\,X(N) = A(1, N + 1)$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$A(I, 1)\,X(1) + A(I, 2)\,X(2) + \cdots + A(I, N)\,X(N) = A(I, N + 1)$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$A(N, 1)\,X(1) + \cdots + A(N, N)\,X(N) = A(N, N + 1)$$

1º Se determina X(1) por el valor de otras incógnitas en la primera ecuación. Se sustituye en las ecuaciones siguientes por su valor en función de las otras incógnitas extraídas de la primera ecuación: multiplicando la primera línea por A(I, 1) y la línea-I por A(1, 1) y restándolas se obtiene una ecuación sin X(1). Se elimina entonces X(1) de las ecuaciones 2 a N, teniéndose así un sistema de N-1 incógnitas (X(2), X(3), ..., X(N-1)) y una ecuación que calcula X(1) en función de las anteriores.
Se itera el procedimiento con el nuevo sistema, disminuyéndose así el orden del sistema de 1 en 1; al llegar al orden 1 se tiene una ecuación de primer grado con incógnita X(N).

2º Con lo anterior se tiene un sistema "triangular". Resolviendo la última ecuación se determina X(N). Luego se sustituye este valor en las ecuaciones 1 a (N-1), obteniéndose un sistema triangular de orden N-1 en el que la última ecuación es una ecuación de primer grado en N-1.
Se repite el procedimiento calculando sucesivamente todas las raíces.

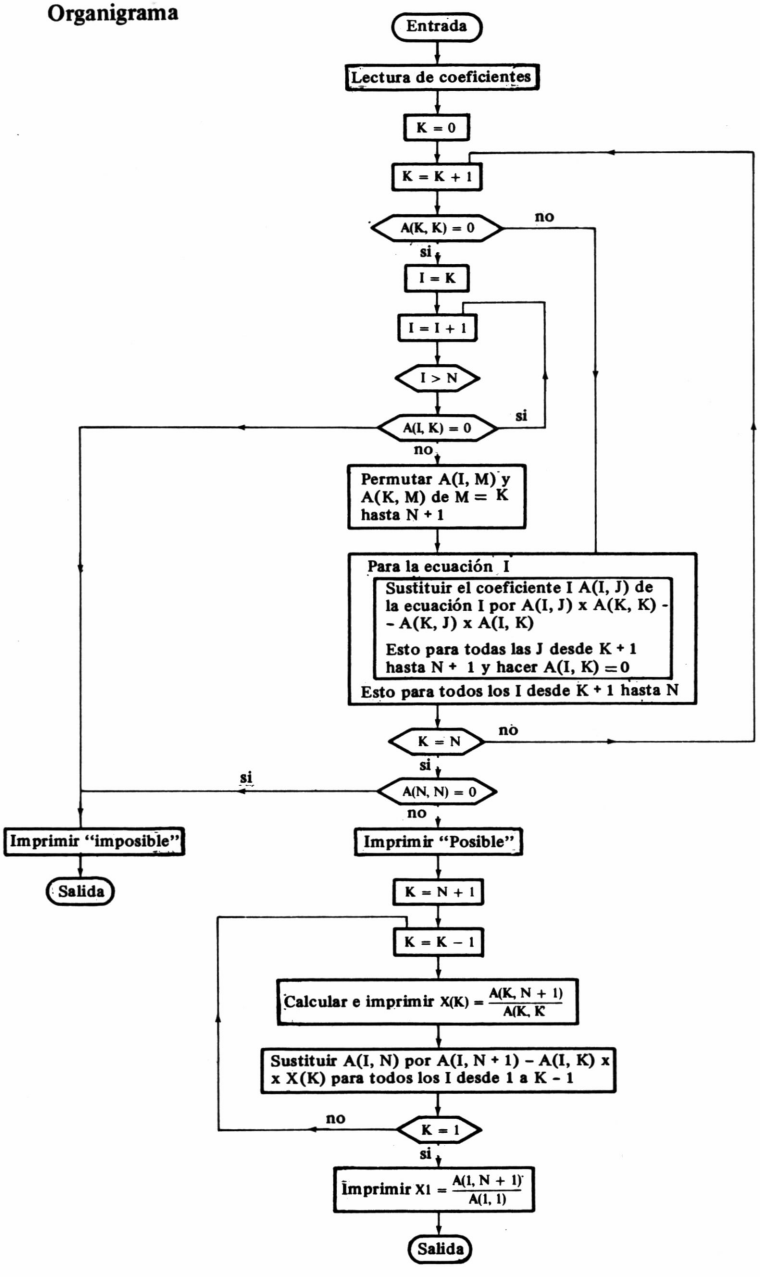
3º Hay una imposibilidad de funcionamiento de este algoritmo en el caso en que aparezca una ecuación de primer grado de la forma

$$0 * X(I) = A(I, N + 1),$$

o sea si el coeficiente A(I, I) = 0; si sucede esto, se busca la ecuación siguiente que tenga el coeficiente de X(I) diferente de cero: si no se encuentra el sistema es imposible, pero si se halla, se permutan las dos ecuaciones.
Si sucediera esto en la iteración N entonces ya no hay más ecuaciones a permutar y el sistema es imposible.

Nota: para realizar una permutación entre los valores de dos variables hay que usar una tercera variable intermedia.

Organigrama



Programa

```
 10  READ N
 20  DIM A(4, 5)
 30  FOR I = 1 TO 4 : FOR J = 1 TO 5
 40  READ A(I, J) : NEXT J : NEXT I
 50  FØR K = 1 TØ N − 1
 60  IF A(K, K) < > 0 THEN 170
 70  FØR I = K + 1 TØ N
 80  IF A(I, K) < > 0 THEN 110
 90  NEXT I
100  GØTØ 380
110    FØR M = K TØ N + 1
120    LET B = A(I, M)
130    LET A(I, M) = A(K, M)
140    LET A(K, M) = B
150    NEXT M
160
170    FØR I = K + 1 TØ N
180        FØR J = K + 1 TØ N + 1
190        LET A(I, J)=A(I, J) * A(K, K)−A(K, J) * A(I, K)
200        NEXT J
210    LET A(I, K) = 0
220    NEXT I
230  NEXT K
240  IF A(N, N) = 0 THEN 380
250
260  PRINT "SISTEMA POSIBLE, LAS SOLUCIONES SON:"
270  FØR K = N TØ 2 STEP − 1
280  LET B = A(K, N + 1)/A(K, K)
290  PRINT « X »; K ; « = »; B
300    FØR I = 1 TØ K − 1
310    LET A(I, N + 1) = A(I, N + 1) − A(I, K) * B
320    NEXT I
330  NEXT K
340  LET X1 = A(1, N + 1)/A(1, 1)
350  PRINT « X 1 = »; X1
360  STØP
370
```

```
380  PRINT "SISTEMA IMPOSIBLE"
390  STØP
400  DATA 4
410  DATA 1, 2, - 1, - 2, 0
420  DATA 1, 1, - 1, - 1, 4
430  DATA 1, 3, - 3, - 1, 2
440  DATA 3, 2, 1, 1, 5

RUN

SISTEMA POSIBLE, LAS SOLUCIONES SON:
X 4 =    .142857
X 3 = - 2.85714
X 2 = - 3.85714
X 1 =   5.14286

USED     3.33   SEC.
```

5.5 CALCULO DEL INTERES COMPUESTO

Enunciado

Sea una cantidad S depositada en un banco al interés I. Se pide el valor del capital acumulado al cabo de N años.

Análisis del problema

Se calcula S1 = S * (1 + I) ↑ N

Consultar los párrafos 1 y 2.

5.6 AHORRO DE MEMORIA. BUCLES

Enunciado

Sea el sistema lineal de ecuaciones que ya se tiene en forma triangular:

$$\begin{aligned} a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n &= a_{1,n+1} \\ a_{22} x_2 + \cdots + a_{2n} x_n &= a_{2,n+1} \\ &\text{-----------} \\ a_{nn} x_n &= a_{n,n+1} \,. \end{aligned}$$

cuya resolución puede hacerse de manera idéntica al ejercicion nº 4; pero aquí se quiere ahorrar al máximo la memoria necesaria. El reservar memoria para A(N, N + 1) supondría perder mucho espacio, ya que la mitad de los valores son nulos y no se aprovechan.

Hallar un método para disponer los coeficientes no nulos siguiendo un vector de una dimensión y resolver el sistema con esta representación.

Consultar los párrafos 2.3, 2.4, 2.5 y 2.6.

5.7 CALCULO DE IMPUESTOS

Enunciado

Dado el salario anual S formado por N partes, que se leen con INPUT, calcular el impuesto a pagar sobre el beneficio en el caso general más sencillo.

5.8 METODO DE NEWTON DE RESOLUCION DE UNA ECUACION

Enunciado

Para resolver la ecuación F(x) = 0, el método de Newton, se emplea el hecho que si x_i es una aproximación de la raíz x_0 de la ecuación, entonces

$$x_{i+1} = x_i - \frac{F(x_i)}{F'(x_i)}$$

es una aproximación mejor de x_0 (F'(x) es la derivada de F(x)).

Análisis del problema

Consideremos el ejemplo:

$$F(x) = \text{sen } x - 0.3$$

y buscamos pues los ángulos x, tales que el seno valga 0.3.

En primera aproximación el ángulo está comprendido entre 0 y 1 radián. Tomemos el valor

$$x_1 = 0{,}5\,.$$

sabemos que $F'(x) = \cos x$.

La fórmula de Newton da

$$x_{i+1} = x_i - \frac{\sin x_i - 0{,}3}{\cos x_i}.$$

Se hará un bucle en el programa en el que sustiuirá

$$\text{X} \quad \text{por} \quad \left(\text{X} - \frac{\sin \text{X} - 0{,}3}{\cos \text{X}}\right)$$

y se detendrá el cálculo cuando (sen X − 0.3) < cierta constante dada de error.

Ver los párrafos 2.3 y 2.7.2.

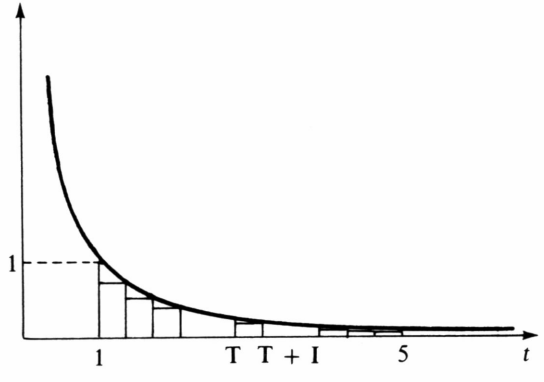
5.9 CALCULO DE UNA INTEGRAL

Enunciado

Calcular

$$\int_1^x \frac{1}{t}\,dt.$$

Nota: se puede calculr el valor de $\int_1^x \frac{1}{t}\,dt$ para $x = 5$ por ejemplo. Se dividirá el eje t en intervalos iguales de ancho I.



y se calculará la suma

$$\sum_{T=1}^{5} \frac{1}{T} * I,$$

aumentando T en I entre un término y el siguiente (o sea T = T + I). Esta suma finita representa la suma de las superficies elementales de la figura: con ella se aproxima

$$\int_{1}^{5} \frac{1}{T} dT,$$

que es la superficie comprendida entre la curva y el eje de las T limitada entre las abcisas T = 1 y T = 5. Se sabe que

$$\int_{1}^{x} \frac{1}{t} dt = \ln x$$

se comparará pues, para un valor de I dado

$$\sum_{T=1}^{5} \frac{1}{T} * I$$

con el valor ln 5, que se puede calcular con la función incorporada o de biblioteca Log(X). Se puede calcular la diferencia entre dos valores anteriores en función de I y se determinará la I que anula esta diferencia con una precisión dada.

Ver los párrafos 2.3 y 2.7.2.

5.10 PLAN DE AHORRO-VIVIENDA

Enunciado

Dada una cantidad de ahorro mensual M y un capital inicial I, calcular el préstamo conseguido al cabo de 4 años, los plazos mensuales de devolución del mismo durante N años y la cantidad total de que se dispone al cabo de 4 años para comprar un apartamento con el sistema o ley "Ahorro-vivienda".

Notas:

1º El interés de las cantidades ahorradas durante los 4 años es del 8 % y su valor total es I_1. El interés para la devolución del préstamo es del 5,5 % y su valor es I_2. La ley especifica que el préstamo concedido al cabo de los 4 años debe ser tal que se cumpla

$$I_2 = 2{,}5 \times I_1$$

2º El capital total ahorrado que se tiene al cabo de los 4 años es la suma de los:

a) las cantidades entregadas en los 4 años:

$$I + (4 \times 12 \times M)$$

b) El interés al 8% de las cantidades entregadas

$$I_1 = (0{,}08 \times 4 \times I) + \sum_{k=0}^{47} M(48 - k)\frac{0{,}08}{12}.$$

(Para calcular esta suma, emplear la relación

$$\sum_{i=1}^{n} (i) = \frac{n(n+1)}{2}\Bigg);$$

c) El valor del préstamo concedido (tal que $I_2 = 2{,}5 \times I_1$).

3º El cálculo de I_2 si se devuelve el préstamo P en cantidades mensuales M_1 durante N años, es tal que

$$M_1 = P\frac{i}{12}\frac{(1+i)^N}{(1+i)^N - 1} \quad \text{(siendo } i = 5{,}5/100\text{)}.$$

Luego $I_2 = (48 \times M_1) - P$.

5.11 HALLAR UNA SECUENCIA DADA DE CARACTERES

Enunciado

Dado un fichero binario, el problema consiste en hallar todos los grupos formados por 5 ceros sucesivos. Emplear el criterio de que antes y después de los 5 ceros debe existir un uno.

Nota:

Hallar un método lo más rápido posible y que conduzca a un programa lo más corto posible. Tener precaución con el fin de fichero.

5.12 PROGRAMACION NO NUMERICA (JUEGO DE DAMAS)

Enunciado

Se tiene una posición dada del juego de damas, es decir se supone que se está en medio del juego (la salida es un caso particular).

El tablero se puede representar con una matriz A(10, 10). Pedir al usuario dos enteros I y J que representan la posición A(I, J) del tablero. Hallar uno de los casos siguientes:

1º no hay peón en dicha posición.

2º el peón es amigo (o enemigo) y puede moverse a (I', J'), (I'' y J'')

3º el peón es amigo (o enemigo) y se "come" los peones situados en

$$(I, J), \quad (I', J'), \ldots$$

o bien en

$$(I_1, J_1), \quad (I'_1, J'_1), \ldots$$

suponiendo que no hay ninguna dama en el tablero.

Análisis del problema

Se puede representar que una casilla está vacía con A(I, J) = 0, que tiene un peón amigo con A(I, J) = 1 y si el peón es enemigo con A(I, J) = = -1.

El caso 1º no presenta dificultad. Debemos comprobar que los números leídos I, J son enteros comprendidos entre 1 y 10.

El caso 2º es interesante. Ir con cuidado con los bordes del tablero.

El caso 3º es difícil y necesitará varios bucles anidados.

En programación no numérica se emplean mucho los "recorridos de un árbol". En este problema se puede suponer que un peón no puede comer más de cinco peones y que un peón no puede tener más de cuatro maneras diferentes de comer ... Si se superan estos límites se impri-

me un mensaje del tipo: "Vd. dispersa demasiado sus peones, juego no interesante ...".

Ver el párrafo 2.6.

5.13 UN METODO PARA GENERAR NUMEROS PRIMOS DE LA SERIE DE FIBONACCI

Enunciado

Obtener una serie de números primos a partir de la serie definida por:

$$F_i = F_{i-1} + F_{i-2}$$

siendo F_i el término i-simo de la serie obtenida partiendo de $F_1 = F_2 = 1$ y llamada serie de Fibonacci.

Método

Un número se dice que es primo si no es divisible más que por él mismo y la unidad; por ejemplo 4 no es primo, ya que sus divisiones son 1, 2 y 4. En cambio 5 es primo puesto que no tiene más división que 1 y 5. Para saber si un entero F es primo se puede mirar si existe un Q tal que

Q = F/J y si Q1 = INT(Q) (parte entera de Q)

entonces es necesario que Q = Q1.

Programa Basic

```
10  PRINT « N = » : INPUT N
20  PRINT
30  LET F1 = 1 : LET F2 = 1
40  PRINT "SUCESION DE LOS N PRIMEROS NUMEROS
    PRIMOS DE LA SERIE"
45  PRINT « DE FIBONACCI »
50  PRINT 1
60  PRINT 2
70  FOR I = 3 TO N
80  F = F1 + F2
90  FOR J = 2 TO F - 1
100 Q = F/J
```

```
110  Q1 = INT(Q)
120  IF Q = Q1 THEN 160
130  NEXT J
140  PRINT F
150  GOTO 170
160  F2 = F1
170  F1 = F
180  NEXT I
190  END
```

Resultados

```
N = ? 30
SUCESION DE LOS N PRIMEROS NUMEROS PRIMOS DE LA
SERIE DE FIBONNACCI
    1
    2
    3
    5
   13
   89
  233
 1597
28657
514229
```

5.14 FUNCIONES FACTORIALES. NUMERO DE VARIACIONES Y DE COMBINACIONES

Enunciado

Determinar el valor de N!, luego el número de variaciones de N objetos tomados de P en P y también el número de combinaciones de N objetos tomados de P en P.

Método

Se sabe que N! = 1 x 2 x ... x N.

El número de variaciones V_n^p de n objetos tomados de p en p viene dada por

$$V_n^p = n(n-1)(n-2)\ldots(n-p+1)$$

es decir

$$V_n^p = \frac{n!}{(n-p)!}.$$

Por último el número de combinaciones de n objetos tomados de p en p viene dado por

$$C_n^p = \frac{n!}{p!\,(n-p)!} = \frac{V_n^p}{p!}.$$

Programa 1

```
10  PRINT "DAR LOS VALORES DE N Y P"
20  INPUT N, P
30  A = 1
40  FOR T = 1 TO N
50  A = A * T
60  NEXT T
70  PRINT "FACTORIAL", N, " = ", .A
80  K = N - P
90  B = 1
100 FOR I = 1 TO K
110 B = B * I
120 NEXT I
130 V = A/B
140 PRINT "VARIACIONES DE", N, "OBJETOS", P,
    « A », P, « = », V
150 D = 1
160 FOR J = 1 TO P
170 D = D * J
180 NEXT J
190 C = V/D
200 PRINT "COMBINACIONES DE", N, "OBJETOS", P,
    « A », P, « = », C
210 END
```

Programa 2

Es más ventajoso emplear el concepto de subprograma. Sea el subprograma de compilación separada

```
      FAC
10    A = 1
20    FOR I = 1 TO M
30    A = A * I
40    NEXT I
50    RETURN
```

Programa principal

```
10    PRINT "DAR LOS VALORES DE N Y P"
20    INPUT N, P
30    M = N
40    CALL FAC
50    PRINT "FACTORIAL", N, " = ", A
60    B = A : M = N - P
70    CALL FAC
80    V = B/A
90    PRINT "VARIACIONES DE", N, "OBJETOS", P,
      « A », P, « = », V
100   M = P
110   CALL FAC
120   PRINT "COMBINACIONES DE", N, "OBJETOS", P,
      « A », P, « = », V/A
130   END
```

5.15 SIMULACION DE UNA LOTERIA

Enunciado

Se trata de realizar una extracción o tirada aleatoria de 7 números, comprendidos todos ellos entre 1 y 49 ambos inclusive.

Método

Se empleará la función de biblioteca RND que suministra un número al azar comprendido entre 0 y 1. Como esta función es pseudo-aleatoria se comprobará que el mismo número no aparece varias veces en una misma tirada.

Programa

```
10  DIM A(7)
20  PRINT "QUIERE REALIZAR UNA TIRADA"
30  INPUT C$
40  IF C$ = "NO" THEN 140
50  PRINT "TIRADA DE LA SEMANA"
60  FOR J = 1 TO 7
70  A(J) = INT (49 * RND(25) + 1)
80  NEXT J
90  FOR I = 1 TO 6
95  IF A(I) < > A(I + 1) THEN 110
100 A(I + 1) = INT(49 * RND(25) + 1)
105 GOTO 95
110 NEXT I
120 FOR I = 1 TO 6 : PRINT A(I) : NEXT I
130 PRINT "NO. COMPLEMENTARIO:", A(7)
140 PRINT "HASTA LA PROXIMA VEZ
150 END
```

APENDICE 1

Recapitulación de las instrucciones de Basic

Instrucción	Ejemplo de uso	Ver pág.
DATA	10 DATA "DATOS", 30, –20, 4, 3	31,32
DEF	10 DEF FNT(A) = (49 * RND(1) + 1)	44
DIM	10 DIM A(5, 10), B(30), C$(15)	38
END	100 END	34
FOR... TO...	10 FOR I = 1 TO 30 STEP 5	36 - 44
:	:	
:	:	
NEXT	100 NEXT I	
GOTO	10 GOTO 150	30
GOSUB	10 GOSUB 300	47
IF... THEN	10 IF A > = B THEN 100	33
INPUT	10 INPUT A, C$	30
LET	10 LET X = (– B + SQR(D))/2 * A	26
PRINT	10 PRINT « X = »; X, Y(J)	28
READ	10 READ K, A(I), B(I, J), N$, M$(I)	26
RETURN	10 RETURN	49
STOP	1000 STOP	41

Extensiones del Basic

CHAIN	17000	CHAIN "ORIGEN" 31000	71
FNEND	40	FNEND	
INPUT LINE #	10	INPUT LINE # 1, A$	75
IF-THEN-ELSE	10	IF A + B = 0 THEN 120 ELSE A + B = K	62
ON ERROR-GOTO	10	ON ERROR GOTO 32700	68
ON-GOTO	10	ON X GOTO 100, 110, 130	61
ON-GOSUB	10	ON K GOSUB 100, 150	61
FOR-STEP-WHILE	10	FOR X = 1 STEP2 WHILE Z < 30	65
FOR-STEP-UNTIL	10	FOR X = 1 STEP3 UNTIL Z > = 30	65

Gestión de ficheros

Instrucción	Ejemplo de uso	Ver pág.
FIELD #	100 FIELD # 10 %, 30 % AS N$	78
GET #	100 GET # 3, RECORD I % FOR I % = 1 % TO 100 % STEP 5 %	77
CLOSE	100 CLOSE 1 %, 7 %	74
OPEN	100 OPEN « TOTO » AS FILE 2 %	73
PUT #	100 PUT # N % RECORD 254 %	78
DIM #	15 DIM # 10, Z$ (200 %), ...	84

APENDICE 2

Las funciones incorporadas o de biblioteca

FUNCION	EJEMPLO
ABS	10 LET X = ABS(Y)
ATN	10 LET Y = ATN(X)
ASC	10 LET N = ASC(M)
CHR$	10 LET N$ = CHR$(N)
COS	10 LET Y = COS(X)
COT	10 LET Y = COT(X)
EXP	10 LET Y = EXP(X)
INT	10 LET Y = INT(X)
LOG	10 LET Y = LOG(X)
RND	10 LET Y = RND(X) siendo X un entero cualquiera
SIN	10 LET Y = SIN(X)
SQR	10 LET Y = SQR(X)
TAB	10 PRINT TAB(N) ; X
TAN	10 LET Y = TAN(X)

Indice alfabético

INDICE ALFABETICO

Otros libros de INFORMATICA publicados por PARANINFO S.A.

Generalidades

ABRAMSON.– Teoría de la información y codificación (5ª edición). 1981.
FLORES.– Estructuración y proceso de datos (4ª ed.). 1984.
GARCIA SANTESMASES y otros.– Cibernética. Aspectos y tendencias actuales. 1980.
GOSLING.– Códigos para ordenadores y microprocesadores. 1981.
LEWIS y SMITH.– Estructuras de datos. Programación y aplicaciones. 1985.
NANIA.– Diccionario de Informática. 1985.
OLIVETTI.– Diccionario de Informática. Inglés-Español (6ª edición). 1985.
PUJOLLE.– Telemática. 1985.
SCHMIDT y MEYERS.– Introducción a los ordenadores y al proceso de datos (5ª edición). 1985.
URMAIEV.– Calculadores analógicos. Elementos de simulación. 1983.

Hardware (Equipo Físico)

ANGULO, J. Mª.– Electrónica digital moderna (5ª ed.). 1984.
ANGULO.– Memorias de burbujas magnéticas. 1982.
ANGULO.– Microprocesadores. Arquitectura, programación y desarrollo de sistemas. (3ª edición). 1984.
ANGULO.– Microprocesadores. Curso sobre aplicaciones en sistemas industriales (4ª edición). 1985.
ANGULO.– Microprocesadores. Fundamentos, diseño y aplicaciones en la industria y en los microcomputadores. (4ª edición). 1985.
ANGULO.– Microprocesadores. Diseño práctico de sistemas. 1985 (2ª edición).
ANGULO.– Microprocesadores de 16 bits. El 68000 y el 8086/8088. 1984.
GARLAND.– Diseño de sistemas microprocesadores. (2ª edición). 1984.
HALSALL y LISTER.– Fundamentos de microprocesadores. 1984.
ROBIN y MAURIN.– Interconexión de microprocesadores. (2ª edición). 1985.
RONY.– El microprocesador 8080 y sus interfases. 1984.

Lenguajes

BELLIDO y SANCHEZ.– BASIC para maestros. (2ª ed.). 1985.
CHECROUN.– BASIC. Programación de microordenadores. (5ª edición). 1985.
DELANOY.– Ficheros en BASIC. (2ª edición). 1985.
GALAN PASCUAL.– Programación con el lenguaje COBOL. (3ª edición). 1985.
GARCIAMERAYO.–Programación en FORTRAN 77. 1985.
HART.– Diccionario del BASIC.
LARRECHE.– BASIC. Introducción a la programación. (5ª edición). 1985.
MARSHALL.– Lenguajes de programación para micros. 1985.
MONTEIL.– Primeros pasos en LOGO. 1985 (2ª edición).
ROSSI.– BASIC. Curso acelerado (4ª edición). 1985.
SANCHIS LLORCA, F.J. y MORALES LOZANO, A.– Programación con el lenguaje PASCAL (5ª ed.). 1984.
WATT y MANGADA.– BASIC para niños. (5ª ed.). 1985.
WATT y MANGADA.– BASIC avanzado para niños. (3ª edición). 1985.
WATT y MANGADA.– BASIC para niños con el microordenador DRAGON. 1985.

Aplicaciones e Informática Profesional

ANGULO.– Curso de Robótica. (2ª edición). 1985.
ANGULO.– Robótica práctica. 1985.
ANGULO.– Prácticas de microelectrónica y microinformática. (2ª edición). 1985.
ASPINALL.– El microprocesador y sus aplicaciones. 1984.
BANKS.– Microordenadores. Cómo funcionan, para qué sirven. 1984.
BELLIDO.– Amaestra tu DRAGON. Curso de programación en Basic para el Microordenador DRAGON. 1985.
BELLIDO.– Cómo programar su Spectrum. (7ª ed.). 1985.
BELLIDO.– Cómo usar los colores y los gráficos en el Spectrum (3ª edición). 1984.
Con casete incorporado.
BELLIDO.– KIT de gráficos para Spectrum. 1984.
Juego completo.
BELLIDO.– Enciclopedia del Spectrum. Tomo I. 1985.
BELLIDO.– Spectrum. Iniciación al Código Máquina. 1985.
BELLIDO.– ZX81. Curso de programación BASIC. 3ª ed. 1984.
ELLERSHAW.– Las primeras 15 horas con el Spectrum. (2ª edición). 1985.
ERSKINE.– Los mejores programas para el ZX Spectrum. 1985.
ESCUDERO.– (Centro de Investigación UAM-IBM).– Reconocimiento de patrones (Fundamentos teóricos, algoritmos y aplicaciones de la moderna técnica denominada "Pattern Recognition"). 1977.
FERRER.– Programas en Basic. 1985.
GAUTHIER y PONTO.– Diseño de programas para sistemas. (4ª edición). 1985.
HARTMAN, MATTHES y PROEME.– Manual de los sistemas de información:
Parte 1 (7ª edición). 1985.
Parte 2 (7ª edición). 1985.
LEPAPE.– Programación del Z80 con ensamblador. 1985.
LUCAS, Jr.– Sistemas de información. Análisis. Diseño. Puesta a punto. 1984.
MARTINEZ VELARDE.– El libro de código Máquina del Spectrum. (2ª edición). 1985.
MONTEIL.– Cómo programar su Commodore-64. Tomo I. Basic. Gráficos. Sonidos. (2ª edición). 1985.
PANNELL, JACKSON y LUCAS.– El microordenador en la pequeña empresa. 1984.
PLOUIN.– IBM-PC. Características. Programación. Manejo. (2ª edición). 1985.
QUANEAUX.– Tratamiento de textos con Basic. 1985.
WILLIAMS.– Programación paso a paso con el Spectrum. (2ª edición). 1985.

EDITORIAL PARANINFO S.A.
Magallanes, 25 - 28015 Madrid

ISBN: 84-283-1244-3