

ENCICLOPEDIA PRACTICA DE LA
INFORMÁTICA
APLICADA

12

APL: Lenguaje
para programadores
diferentes

Juan Ruiz de Torres

Director-Editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO,
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicacion
y Licenciado en Informática
JOSE ARTECHE, Ingeniero de Telecomunicación

Diseño y maquetación:
BRAVO-LOFISH.

Dibujos:
JOSE OCHOA Y ANTONIO PERERA

Tomo XII. APL. Lenguaje para programadores diferentes.
JUAN RUIZ DE TORRES, Dr. Ingeniero Industrial

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Sor Angelia de la Cruz, 24-7.^o G. Telef. 279 40 36. 28020 Madrid.

Publicidad:

Golfar Publicidad, S.A. Benito de Castro, 12 bis. 28020 Madrid.
Distribución en España:

COEDIS, S.A. Valencia. 245. Telef. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Telef. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América. 1532. Telef.: 21 24 64.
Buenos Aires. 1.290. Argentina.

Todos los derechos reservados. Este libro no puede ser, en parte o totalmente, reproducido, memorizado en sistemas de archivo, o transmitido en cualquier forma o medio, electrónico, mecánico, fotocopia o cualquier otro, sin la previa autorización del editor.

ISBN del tomo: 84-7688-044-8.
ISBN de la obra: 84-7688-018-9.

Fotocomposición:

ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Impime:
MATEU CROMO Pinto (Madrid).

Ediciones Siglo Cultural, S. A., 1986

Depósito legal: M-42.836-1986

Printed in Spain - Impreso en España

Rescripciones y números atrasados:

Ediciones Siglo Cultural, S.A.

Sor Angelia de la Cruz, 24-7.^o G. Telef. 279 40 36. 28020 Madrid

Ambre, 1986.

P. V. P. Canarias: 365.

INDICE

- 1 Ni inglés ni español.—Hacia el lenguaje natural.—¿Por qué tantos lenguajes de programación?—Un canadiense, las cuatro reglas y el cangrejo.—Operadores sin título de medicina.—Argumentos sin película.—Las variables, cajones para usos múltiples.—Interpretar y compilar.—Llega el vector. 9
- 2 Un operador «muy potente».—La tabla lisa y la hipermatriz cubica: entremos en la cuarta dimensión... y más allá.—Un escalar que no tiene que ver con montañas.—El sañire del APL: mide y confecciona. 19
- 3 El sañire del APL.—Ferretería y cerebretaria.—¿Y donde trabajo?—Monogamia y poligamia.—Sin equivocarse.—Eee... sin hacer final. 29
- 4 A la «conversión» por el milagro. El bonito caso de los intereses al capital invertido, y otros problemas del que maneja dinero.—Porcentajes.—Beneficios y reparto de ganancias en el mus.—Promedios. 37
- 5 Otra letra griega.—El tren de los números naturales.—El operador de la suerte: ¿no es para preguntárselo dos veces?—Extendamos las operaciones a la tribu completa.—Una solución a la lotería por el APL.—La prueba del teléfono.—Y las letras, ¿quién las maneja? 45
- 6 La jota bigama y la h perdida en el tumulto.—Tablas de nombres.—Una coma mágica: pega variables o las aplasta.—Haciendo vectores de los escalares. 55

7	La delia al revés y el APL servicial.—Corregir sobre la marcha.—La cesta de guindas.—Haciendo sencilla a Su Majestad la Estadística.—¿Qué pasa si se va la luz?	63
8	Del caño al coro y del coro al caño.—Las propiedades singulares de la T, del derecho y del revés.—Barriendo la casa.	75
9	Cómo colar a Pepe en la fila del cine.—Un paréntesis poco redondo.—Ordenando lo que está en desorden.—El primero y el último de la clase.—¿Quién se va de viaje?—Moviéndose por la función.—Compresión viene de comprimir.—Variables limpias.	83
10	Dialoguemos con el ordenador por la ventana.—Un zahorí algo trampo.—El psiquiatra en el ordenador. Siempre con el ejemplo.	95
11	Sigamos con las cosas lógicas: operaciones booleanas (que meten miedo).—Los soldados que pasan debajo de la viga.—Más difícil todavía: los soldados de permiso (los cinco primeros y los cinco últimos).—Por el techo y por el suelo.	103
12	Programa para el cálculo de los valores dietéticos de una receta culinaria.	113
13	Programa para crear el fichero de calificaciones en una clase.	119
	Conclusión.	131
	Apéndices.	133

*A Ken Iverson, Manuel Alfonseca
y María Luisa Tavera*

PRESENTACION

E

S siempre temible el autor que asegura a sus lectores que su libro será «diferente» en algún aspecto. *Nihil novum sub sole*, decían los antiguos. Así que el título de este libro puede parecer pretencioso.

EL LENGUAJE APL: ¿ES DIFERENTE?

Y, sin embargo, es la firme convicción, no ya del propio autor, que no tendría mucha importancia, sino de cuantos han conocido el lenguaje APL, que es una notable contribución al saber de los «informáticos», así como de los no iniciados. Por dos razones: por ser un lenguaje de programación potente y fácil de aprender (de sus limitaciones ya hablaremos); y por tratarse al mismo tiempo —y esto no es tan común— de una elegantísima herramienta de «pensar».

Este libro tratará de demostrar ambas capacidades del APL. Lo hará siguiendo una premisa que se fija el autor: los lectores pueden ser personas ya cultas en los arcanos de la Informática, pero también tienen derecho a seguirlo y entenderlo los novicios en esa nueva ciencia humana. Por ello, el lenguaje y los ejemplos seguidamente presentados se han elegido sin abandonar el rigor, pero sin complicarlo bajo capas de nebla oscuradora, que a veces se invocan como «necesarias» en un libro «científico».

¿LIBRO TECNICO O DE DIVULGACION?

Por otra parte, es conveniente advertir también al lector que no se trata de un texto de APL; los hay excelentes ya (véase la Bibliografía) y no pretendemos ocupar su puesto. Por el contrario, lo que este pequeño tra-

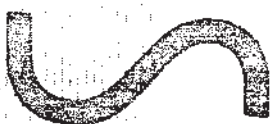
tado pretende *desmitificar* el APL ante aquellos que lo creen difícil o exótico, y despertar el interés de quienes, sin haber oído hablar de él, quieren saber por qué hay un libro en esa Biblioteca con ese título: *APL: LENGUAJE PARA PROGRAMADORES DIFERENTES*.

¿LIBRO «MISIONERO»?

Quizá surja entre muchos de los lectores el convencimiento, tras la lectura de este libro de que **ellos** son «diferentes», porque el APL les ha intrigado y cautivado; quizá esos potenciales programadores «diferentes» sigan el difícil curso que les convierta en expertos, o al menos, en usuarios del APL. Esa habrá sido una buena compensación al esfuerzo del autor.

Por último, cabe añadir que en este volumen no se tratan a fondo todos los aspectos relacionados con el APL. En bien de la brevedad, y para no exagerar la longitud del texto, no se han tratado varias áreas: ficheros y variables comparadas, funciones matemáticas y circulares, tratamiento de pantallas y otros periféricos, comunicaciones, enlace con otros lenguajes. Quizá, si el interés de los lectores lo justifica, haga el autor una excursión futura por esas otras tierras. Por el momento, lo que en los capítulos que siguen se expone debería ser bastante para convencer a los lectores de lo «diferente» que es el lenguaje que se presenta: el APL.

Ni inglés ni español.—Hacia el lenguaje natural ¿Por qué tantos lenguajes de programación?—Un canadiense, las 4 reglas y el cangrejo.—Operadores sin título de medicina.—Argumentos sin película.—Las variables, cajones para usos múltiples.—Interpretar y compilar.—Llega el vector.



El lector se ha acercado a otros lenguajes de programación, habrá habitualmente encontrado que su «sintaxis» se basa en ciertas reglas para unir los «comandos» u órdenes a la máquina. Los comandos son en general palabras inglesas como goto, add, move, if, etc. Esto es: los lenguajes de programación tratan de resolver los problemas, dando a los usuarios una herramienta «cercana» al lenguaje natural, esto es, al inglés... de los programadores de habla inglesa.

¿HABLARLE EN ESPAÑOL AL ORDENADOR?

Por supuesto, ningún lenguaje de programación se acerca —por ahora— al auténtico lenguaje «natural» de los usuarios. Por eso, es una falacia pensar que usar palabras de la lengua común —aunque sean en inglés para anglohablantes— va a hacer sencillo programar. Lo ideal sería, claro, poder decirle al ordenador: «Oye, búscame en el fichero de clientes los que deben más de cien mil pesetas, sácame una lista y mándales también una carta un poco conminatoria para que paguen de una vez. ¡Ah!, lo haces en papel timbrado de la empresa. Y date prisa, por favor. SI NO ENTENDISTE ALGO, PREGUNTA.»

No; esto aún no se puede hacer. No porque sea muy difícil para un ordenador, sino por la imprecisión de nuestra lengua. Aun así, la cláusula final debería servir para que el ordenador «preguntase» todos los puntos **no excluyentes** del encargo, como, por ejemplo:

- donde, en qué dispositivo, está ese fichero de clientes;
- qué modelo de carta hay que enviar (si hay varios archivados);
- ¿hay que preparar el sobre también?

etcétera. Todo esto supuesto que el ordenador pudiese hacer el análisis semántico, de contenido, del mensaje recibido, para conocer las cláusulas de mandato inequívocas y eliminar el «ruido» de la información recibida, esto es, las palabras inútiles que no añaden contenido: «Oyes», «en el», «tambien», etc. (se puede, por cierto, argumentar si incluso esas palabras son auténtico «ruido» o deben considerarse «modificativas» del mensaje).

En todo caso, no es ésta la situación, hoy por hoy. Y, a pesar de los esfuerzos que se hacen para construir un lenguaje de programación «natural» —menos cada día; se va en otras direcciones—, la realidad es que seguramente no hace falta ese lenguaje «natural». Los que hay disponibles, una vez pasada la barrera de su aprendizaje, son fáciles de manejar y permiten resolver los problemas que se presentan.

¿POR QUÉ EL APL?

Entonces, ¿para qué otro lenguaje, el APL? En realidad, ¿para qué varios lenguajes de programación? ¿Por qué no uno solo, pero bueno? La respuesta —que, por cierto, no es obvia— es que cada lenguaje es útil en un entorno o grupo de condiciones distinto. Así, el **COBOL**, el **RPG**, son excelentes para resolver problemas de tipo comercial en máquinas de tamaño medio y con diversos periféricos y ficheros; el **BASIC**, para pequeñas máquinas y problemas de escasa complejidad de entrada y salida de información; el **FORTRAN**, para cálculo matemático en ingeniería, aplicaciones científicas y estadística.

KEN IVERSON

¿Y el APL? ¿Nos vas por fin a decir algo del APL?, preguntarán algunos lectores, que comienzan a exasperarse con tanto rodeo. Todo lo anterior es, en efecto, antesala para lo que nos ocupa: la presentación del lenguaje APL. Un profesor canadiense, **Ken Iverson**, inventó una notación basada en operadores matemáticos, para tratar en sus clases y de forma rigurosa las operaciones con conjuntos. Poco más tarde se planteó y resolvió la puesta en práctica de ese «lenguaje» de manejo de «cuernos de elefantes», y nació el APL. «APL», por cierto, quiere decir *A Programming Language*, «un lenguaje de programación»; aquí se observa lo poco pretencioso que fue —y es— Ken Iverson.

El lenguaje APL, pues, no es mucho más que una extensión de la simbología matemática, eso sí, con un rigor peculiar para evitar ambigüedad, doble sentido, en las expresiones construidas.

Por ejemplo (línea a)

$$13 + 5 - 8 - 3 \times 5$$

se puede calcular así en aritmética

$$13 + 5 = 18$$

$$18 - 8 = 10$$

$$10 - 3 = 7$$

$$7 \times 5 = 35$$

o bien

$$13 + 5 - 8 = 10$$

$$3 \times 5 = 15$$

$$10 - 15 = -5$$

Dos resultados distintos.

¿TIENEN RAZON LOS CANGREJOS?

En cambio, la regla fundamental del APL es que siempre se calcularán las operaciones de derecha a izquierda, a no ser que los paréntesis obliguen a cambiar esa prioridad; por eso, la línea a) daría:

$$3 \times 5 = 15$$

$$-3 \times 5 = -15$$

$$8 - 15 = -7$$

$$\begin{array}{r} -8 \\ -13 \\ -17 \\ 5 + 7 = 12 \\ 15 + 12 = 27 \end{array}$$

Sólo si queremos cambiar el orden de la operación escribiremos

$$(13 + 5) - 8 = (3 \times 5) : 10 = 15 : 5$$

Por cierto, para evitar la confusión entre la operación «restar»(-) y el signo negativo, se usa el - para la substracción y los números negativos con un - en alto, así:

$$-3$$

Restar el número 10 del negativo 6 se escribirá:

$$\begin{array}{r} -6 \\ -10 \\ -15 \end{array}$$

O bien:

$$\begin{array}{r} -5 \\ -3 + -2 \\ -5 \end{array}$$

con lo cual hemos eliminado de un plumazo aquella dificultad del signo y la operación.

LOS OPERADORES

En APL, pues, encontraremos «operadores» matemáticos (también llamados «primitivos»), y no palabras del lenguaje (salvo para ciertas operaciones de tratamiento de ficheros, etc). Esos **operadores** son muchos (ver en el Apéndice la tabla completa); es más, la mayor parte de ellos tienen doble uso, como veremos luego.

De esta forma, gozaremos de un riquísimo acervo de posibilidades de acción sobre los elementos a usar. Pero no se suponga que ello obligará a estudiarse a fondo todos los operadores antes de poder programar. Por el contrario, usando muy pocos de ellos podemos realizar programas incluso potentes. Recuerde el lector que, salvo en contadas ocasiones, para la vida diaria (no se usan más que las cuatro operaciones aritméticas).

En APL, por tanto, tendrán mayor elegancia y posibilidades los programas cuantos más operadores usen; pero con muy pocos ya se podrá, en general, resolver aquel problema, claro que no tan potente o elegantemente.

Vamos, pues, a examinar los más sencillos de esos **operadores**, y a ponerles a prueba en casos prácticos.

OPERADORES BIGAMOS

Ya hemos visto el + y el -. Si, esos son dos operadores importantes en APL. ¿Cómo funcionan? Simplemente, como en la aritmética tradicional: lo que está a un lado se suma (o se resta) a lo que está al otro; esto es, al **argumento** de la izquierda se le **aplica** el operador con el **argumento** de la derecha. Así:

$$\begin{array}{r} 3 \\ + \\ 5 \\ -2 \end{array}$$

y con el operador -

$$\begin{array}{r} 4 \\ - \\ 6 \\ -10 \end{array}$$

El operador \times funciona también de igual forma:

```
I * D
```

(el argumento de la izquierda se multiplica por el argumento de la derecha para hallar el resultado).

Si I vale 10 y D vale 4,

```
I * D
```

```
40
```

CAJONES PARA TODO USO

Observemos que hemos «llamado» I al valor 10; esto quiere decir que I es una **variable**: un **cajón** donde se guardan valores para usarlos más tarde. Si queremos cambiar el contenido del cajón, le asignaremos otro valor con el signo \leftarrow , que se lee «asignar a». Así:

```
I ← 14
```

se leerá: «asígnese el valor 14 a la variable I»; desde este momento I tiene el valor 14. Para conocer el valor de una variable en un momento dado, basta con «declararlo»; esto es, teclearlo en el ordenador y «ejecutar» esa declaración con la tecla «entrada»:

```
I
```

```
14
```

El ordenador responde con el valor de I en ese momento).

Pero ¿cómo hace el usuario del ordenador para poder llegar a «declarar» cosas, cómo hacer operaciones, asignar valores a variables, pedir valor de una variable?

Evidentemente, quien haya usado un ordenador sabe que, si se le enciende y uno tecldea cosas como las de antes, no ocurrirá lo que hemos indicado. Eso es porque antes el **ordenador debe tener en memoria el APL**. Esto es, ha de «cargarse» primero, antes de hacer operaciones, antes de «programar», el **intérprete APL**.

¿UNA CALCULADORA UN POCO CARA?

¿Qué es un intérprete? Es, en palabras pobres, lo que su significado ordinario dice: un artificio, un elemento por el cual la máquina puede entender, interpretar en el lenguaje de la propia máquina (el lenguaje de máquina) lo que le comunicamos en el lenguaje APL. Así:

```
3 * 6
```

que no está en lenguaje de máquina, por muy raro que parezca al usuario primerizo, pero que entiende muy bien quien haya estudiado algo de BASIC (por ejemplo), está en lenguaje APL. Pues bien: si la máquina tiene «cargado» el intérprete APL, esa expresión será traducida, interpretada, a la máquina, que hará la operación con los dos argumentos 3 y 6, obtendrá el resultado 18, y a su vez el intérprete se lo comunicará a la pantalla, donde aparecerá:

```
18
```

¿Esto parece el «huevo de Colón»? Pues, aunque parezca extraño, en lenguajes como el BASIC eso no se puede hacer. Claro que —dirá un lector— eso lo hago con una calculadora corriente.

¡QUE VIENEN LOS VECTORES!

¿Sí? Bueno, pues hagámoslo más complicado, pero siempre, al parecer, elemental:

```
R + 3 4 5
```


Esto es, asignemos a R tres valores; 3, 4 y 5 (que son los radios de tres círculos, por ejemplo).

La longitud de sus circunferencias serían:

$$6,28 \cdot R$$

$$18,8 \quad 25,1 \quad 31,4$$

¿Qué ha ocurrido? El ordenador ha multiplicado el valor de 2π (6,28) por R; no le ha preocupado en absoluto que R fuese «dentro» tres valores (tres radios); ha multiplicado el argumento izquierdo, 6,28, por el derecho R (3 4 5); esto es, ha multiplicado 1 número por 3 números, por un vector, o serie, de tres números. Y ha dado como resultado, lógicamente, un vector de tres números.

¿Lo harías, querido lector, con tu calculadora, así?

• (Por cierto, hay que advertir que el APL trabaja con los números decimales en la notación anglosajona, esto es, la coma decimal se escribe punto decimal; la coma tiene otros usos, que luego veremos.)

RESUMEN

Va siendo hora de cerrar este capítulo. Pero resumamos antes lo expuesto:

- a) el lenguaje APL es una extensión de las Matemáticas, y usa símbolos, no palabras;
- b) para trabajar con APL no es preciso conocer todos sus símbolos u operadores; con unos pocos basta para resolver problemas difíciles;
- c) se necesita «cargar» en la memoria del ordenador el llamado «intérprete» APL, que va examinando cada instrucción que enviamos y la «traduce» al lenguaje propio de la máquina para su ejecución;
- d) uno puede dar instrucciones directas a la máquina, sin preparar un programa; la máquina, a través del intérprete, la ejecutará directamente, si son factibles en APL;
- e) los operadores ejecutan la operación cuyo valor representan entre los argumentos a su derecha e izquierda (veremos esto con mucho más detalle luego), para obtener el resultado;

f) las operaciones APL se ejecutan de derecha a izquierda, entre pares de instrucciones; así:

$$3 + 5 - 4 \times 2$$

$$5 - 8$$

$$3 + -3$$

$$0$$

g) se pueden «guardar» valores en las llamadas variables, auténticos cajones para todo y repetido uso; así:

a)

$$1 + 3$$

$$3$$

b)

$$1 + 6 \quad 7 \quad 8$$

$$1$$

$$6 \quad 7 \quad 8$$

en a) hemos asignado a I el valor 3; I vale, pues, 3. En b) decidimos, porque queremos, cambiar el valor de I (ya no nos hace falta que valga 3) por el vector de valores 6 7 8; esto es, queremos guardar tres números, y en ese orden, en I. Ahora I vale 6 7 8 (no 6 + 7 + 8, sino los tres números); esto es, I es un vector de números, un conjunto de números. Lo que, por cierto, permite hacer operaciones con ese conjunto; así:

$$5 + I$$

$$11 \quad 12 \quad 13$$

Es decir: 5 se suma a cada uno de los elementos de I.

¿Estamos listos para pasar al capítulo segundo? ¿O lo leemos todo de nuevo?

CAPITULO 2

Un operador «muy potente».—La tabla lisa y la hipermatriz cúbica: entremos en la cuarta dimensión... y más allá.—Un escalar que no tiene que ver con monañas.—El sastre del APL: mide y confecciona.

V

AMOS a conocer algunas herramientas adicionales que el programador APL tiene a su disposición. En primer lugar, completemos el cuadro de los *operadores aritméticos*:

$$5 \quad 4 \quad + \quad 2 \quad = \quad 6$$

$$5 \quad - \quad 2 \quad = \quad 3$$

¿De acuerdo? De igual manera:

$$6 \quad 13 \quad 0 \quad - \quad 2 \quad = \quad 4$$

$$2 \quad 9 \quad - \quad 4 \quad - \quad 6 \quad = \quad 4$$

(Ha estado 4 del vector 6 13 0 -2).

Continuemos con la multiplicación:

$$2 \quad 5 \quad 0 \quad \times \quad 1 \quad 2 \quad 3 \quad = \quad 0$$

$$2 \quad 10 \quad 0$$

y la división:

$$6 \div 2 \quad 6 \div 1 \quad = \quad 3 \quad 1 \quad 6$$

En cambio si pedimos:

```

4 3 1 2 3 1
LENGIN ERROR

```

El ordenador nos envia un mensaje de «error de longitud»: los dos vectores no se pueden sumar, evidentemente.



UN ERROR: ¡HORROR!

La potenciación presenta un caso interesante:

```

4 2 * 2
1 6 4

```

Pero tambien:

```

4 9 * 0,5
2 5

```

Esto es, la elevación a 0,5, o sea, 1/2, es equivalentemente a obtener la raíz cuadrada (como saben los que recuerden sus estudios medios). Así, para obtener la raíz cúbica, elevaremos a la potencia 1/3:

```

27 * 1 + 3
5

```

Pero lo interesante de esto es que el APL permite no sólo elevar a cualquier potencia, **ergo**, extraer cualquier tipo de raíz:

```

52 * 1 + 5
2

```

sino también elevar a un decimal; el intérprete se encarga de encontrar sentido matemático y resultado a la operación, que no es fácil de interpretar en aritmética elemental. Así:

```

34 * 1,45
25,9

```

¿Vale? Interesante, al menos, parece.



LAS TRIBUS DEL APL

Hermos visto que el APL trabaja con grupos de valores; ya examinamos los vectores

```

4 6 3 -2 0

```

que son conjuntos ordenados, de números. Pero, ¿no hay otros conjuntos de números para el APL?

Sí, los que están ordenados en tablas o matrices:

```

2 3 4
0 2 1

```

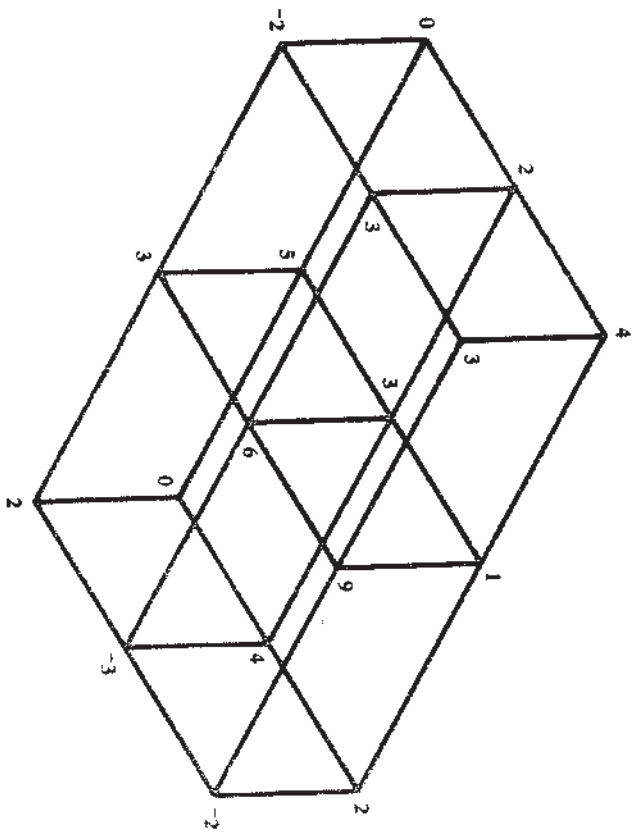
Esta es una tabla de 2 filas y 3 columnas.

HACIA LA CUARTA DIMENSION

No sólo esto, sino que también trabaja con tablas de 3 dimensiones. O de muchas dimensiones. Las de 3 dimensiones se pueden «imaginar» como un paralelepípedo.

Espero que la interpretación del dibujante aclare este cuerpo de números, o hipermatriz, o matriz de 3 dimensiones, que tiene

2«capas»



Y en caja «capa»

así, la segunda capa sería

3 «filas» y
3 «columnas»

3	2	-2
3	6	3
2	3	2

¿Cómo representar una matriz de 4 o más dimensiones? ¡Ay!, para el ser humano, al menos, no es fácil. En cambio, el APL no tiene ningún problema en crearlas y en trabajar con ellas, como veremos a continuación. Y lo que es más, gracias a ello podemos utilizarlas, pues su uso es muy importante en varias ciencias. Pero antes de continuar vamos a recordar de nuevo a nuestros amigos los «cajones para todo uso», las **variables**, y a operar con ellas.

CAJA 1 2 3
CAJITA 1 2

Así, podemos:

	CAJA + CAJITA
-8 -7 -12	
CAJA * CAJITA	
-9 -18 27	
CAJITA + CAJA	
-9 -4.5 3	

Creo que es muy cómodo trabajar así. Claro que hasta ahora hemos trabajado sólo con vectores y con... ¿cómo se llama cuando es un solo número?

DON ESCALAR SIN DIMENSION

Bien: es complicado de recordar, pero importante: son los **escalares** (todo es acostumbrarse). ¿Y en qué se diferencia un **escalar** de un **vector**? La diferencia se ve: un sólo número, o varios números. ¿No? Pues, no. No es bastante. La diferencia está en que el **vector** tiene una **dimensión** (el «largor») mientras el **escalar** es como «un punto» en la serie de los números, o sea, que no tiene dimensiones.

¿Y cómo podemos saber, sin «abrirlo», cuántas dimensiones tiene uno de nuestros «cajones», una variable? Ello es el trabajo que cumple otro operador nuevo, que se representa con el signo ρ (una letra griega llamada «rho», antecesora de nuestra «r»).

UN OPERADOR MONOGAMO: EL SASTRE DEL APL

Así, aplicando el «rho» a una variable, responde:

ρ CAJA	3
-------------	---

Esto es: responde con un número, luego CAJA tiene una dimensión; el número es 3, luego CAJA tiene 3 elementos en esa dimensión: vector con 3 elementos. ¿Y si preguntamos las dimensiones de un **escalar**?

ρ CAJITA	
---------------	--

Pues bien: el ordenador contesta con una línea en blanco, esto es, con 0 números, luego *CALITA* tiene 0 dimensiones; es un **escalar**.

EL VECTOR FANTASMA

¿No debería contestar, entonces, 0? No; ello significaría un número, el 0, o sea, una dimensión con 0 elementos... ¿Eso puede ser? Ya lo creo: **vectores** con 0 elementos se usan en APL con mucha frecuencia (se les llaman **vectores nulos**).

¿Qué pasa con las matrices, o tablas? Pues si imaginamos la tabla

```
3 2 4
1 2 5
```

medida en la variable *CAJON* (ya veremos cómo se hace) y decimos al ordenador:

```
2 3 p.CAJON
```

contesta con 2 números, luego *CAJON* tiene 2 dimensiones (como que es una tabla); los números son 2 y 3; esto es, en la primera dimensión, el «alto», o número de líneas, tiene 2 elementos (2 líneas); en la **segunda** o **ancho**, o número de columnas, hay 3.

Por esta razón, porque «mide» las dimensiones de una variable APL, llamo al «ro» el «sastre» del APL. Un sastre muy útil, pues no sólo «mide» al cliente, sino que le hace los trajes!

Veamos. Imaginemos que queremos «crear» un vector. Podemos usar el *p* así:

```
4 p 5
```

TELA MARINERA

El 4 indica al ordenador las dimensiones (una) y número de elementos que queremos tenga el «traje». Es evidente que vamos a crear un vector

con 4 elementos. Pero, ¿qué elementos? Esos son la «tela» del vector, y son lo que está a la derecha del *p*:

```
5 5 5 5
```

así el vector 5 5 5 5 tiene de «tela» los 4 cincos, y su medida es «4» (una dimensión, un vector; 4 elementos).

Veamos más ejemplos:

```
A + 3 p 4 2 5
A
4 2 5
p A
3
B + 6 p 2 0 3 4
B
2 0 3 4 2 0
p B
6
```

(esto es, cuando se acaba la «tela» toma desde el principio la que «falta»)

EL TRAJE INVISIBLE

¿Qué pasa si la parte izquierda de *p*, el **argumento** izquierdo, es 0?

```
C + 0 p 5
C
```

Evidente: crea un «traje», *C*, con 0 elementos hechos de la «tela» 5. O sea, un *traje sin tela*. ¿Eso quiere decir que *C* no es nada? Apliquémosle el *p*, a ver qué pasar:

```
0 p C
```

Esto quiere decir, según hemos aprendido, que C es un vector (responde con 1 número, aunque sea el 0), con 0 elementos. O sea, un vector nulo. Quizá algún lector piense que un traje, por mucho que se «pueda» medir, si está vacío, es que no existe. No, aseguramos al suspicaz que no se trata del «rey desnudo»; los vectores vacíos, o nulos, son algo distinto de la nada; con ellos trabajaremos y haremos cosas importantes. Por ciento, ¿qué pasa cuando el argumento izquierdo del p es más de un número? Por ejemplo:

$$\begin{matrix} 2 & 3 & p & 5 & 3 \\ 5 & 3 & 5 & & \\ 3 & 5 & 3 & & \end{matrix}$$

El sastrero p ha creado con las medidas 2 3 y la tela 5 3 un traje de 2 dimensiones (una tabla, o matriz), de 2 filas de alto por 3 columnas de ancho.

TRAJES MAS CONSISTENTES

Y puesto que no tenía otra «tela» que 5 3, la ha usado repetidas veces (fila a fila) hasta completar el traje.

Por eso ahora podemos crearos nuestro CAJON de antes:

$$\text{CAJON} = \begin{pmatrix} 2 & 3 & p & 3 & 2 & 4 & 1 & 2 & 5 \end{pmatrix}$$

Es más, podemos también crear:

$$\text{CAJON}^2 = \begin{pmatrix} 2 & 2 & p & 3 & 2 & 4 & 1 & 2 & 5 \end{pmatrix}$$

En CAJON 2 pondremos así una tabla de medidas 2 2 y «tela» hecha con elementos de CAJON:

$$\begin{matrix} 3 & 2 & \text{CAJON}^2 \\ 4 & 1 & \end{matrix}$$

RETALLES

Es evidente que ahora le ha «sobrado tela» (2 5) y no la ha utilizado. En fin, podemos formar una matriz de 3 dimensiones, así:

$$\begin{matrix} 2 & 3 & 4 \\ \text{CAJOTA} & + & 2 & 3 & 4 & p & 0 \\ \text{CAJOTA} & & & & & & \\ \text{CAJOTA} & & & & & & \end{matrix}$$

a)

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

b)

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

¿LA CUARTA DIMENSION?

Así, tenemos las 2 «cajas» (a) y (b), y dentro de cada «caja» 3 filas y 4 columnas. ¿Estamos? Uno puede imaginar bastante bien cómo es CAJOTA, pero, ¿quién puede imaginar el MISTERIO siguiente?

$$\text{MISTERIO} = \begin{pmatrix} 3 & 2 & 5 & 4 & p & 4 & 0 & -3 & 4 \end{pmatrix}$$

Pues bien: MISTERIO tiene 4 dimensiones; es un «hiper» paralelepípedo. Creo que los lectores podrán entender que en la 1.ª dimensión, las columnas, hay 4 elementos; luego hay 5 filas; luego 2 «cajas», y en fin, 3 «hiper-cajas». No se puede ver ni tocar, pero definitivamente se puede trabajar con ello. ¡Vaya MISTERIO!

RESUMEN

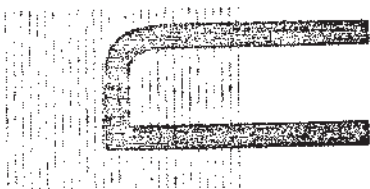
Resumamos lo visto en el capítulo:

— Dos operadores aritméticos se pueden aplicar, no sólo a escalares, o números «sueños», sino a vectores y aún a matrices;

CAPITULO 3

- las variables, «cajas para usos múltiples» permiten almacenar objetos APL, como escalares, vectores, matrices;
- las variables se pueden borrar, guardando en ellas de nuevo otros objetos;
- la «medida» de un objeto APL, esto es, sus dimensiones y número de elementos en cada una de ellas, se hace con el operador ρ (ρ);
- El operador ρ también sirve para «crear» vectores y matrices;
- los escalares **no tienen** dimensiones; los vectores, **una**; las matrices, **más de una** dimensión; puede haber incluso vectores y matrices vacíos.

El sasire del APL.—Ferrertería y cerebería.—¿Y dónde trabajo?— Monogamia y poligamia.—Sin equivocarse.—Eee... sin hace final.



Un punto que no tratamos en el capítulo 2 merece exponerse aquí más ampliamente. Aunque no sea más que para mostrar un poco de «magia» APL. Así, hay que recordar que nuestro «sasire», el ρ , «mide» los objetos APL, y su resultado dice las «dimensiones del objeto». Bien, debemos aclarar que ρ , en forma monádica, esto es, aplicado a un solo argumento, para «medirlo», tiene siempre como resultado un vector:

- si se aplica a un escalar, resultará un vector nulo;
- si se aplica a un vector, resultará un vector con un solo elemento;
- si se aplica a matrices o hipermatrices, resultará un vector de 2, 3, etc., elementos.

HAGAMOS MAGIA

Esto no parece mucha novedad. Pero ¿qué ocurre si lo aplicamos sucesivamente?



(vector nulo, evidentemente).

(porque p, aplicado a un vector nulo, dará un vector de un elemento, el 0, pues el vector nulo no tiene ningún elemento).

1

2 3 4 5

(pues el p del vector 0 deberá ser un vector, de 1 elemento, y ese elemento será el 1, pues ese vector 0 tiene sólo un elemento).

1

2 3 4 5

(ahora siempre dará 1, pues p se aplicará sucesivamente a un vector con elemento 1).

No sé si ha quedado claro; lo importante es fijarse en que p siempre da como resultado un vector.

DONDE TRABAJAMOS

Hasta ahora hemos dado por supuesto que ya teníamos el ordenador encendido, con su APL (su intérprete APL) dentro, y nosotros teclábamos alegremente. Pero ¿cómo hemos llegado hasta aquí? ¿Dónde va lo que teclamos?

Como sabemos, el ordenador electrónico es un conjunto de elementos «físicos»: unidad central, memoria, discos, teclado, impresora... Esto es, un montón de «latas», de piezas de «ferreteria». Y así se le llama en inglés: el *hardware*, palabra que se ha medio incorporado a nuestro diario lenguaje informático.

Este «hardware» no funcionaría si no tuviese además las «instrucciones» que le indiquen qué hacer. Estas instrucciones se denominan, pues no son «latas» o «hierros», no son «hard» (duras), el *software* (ferreteria «blanda»). Suena como una barbaridad idiomática, pero hay que reconocer su extrema sencillez: dice mal, pero dice clarísimo que no es la «parte sólida» del ordenador, sino «palabras», instrucciones, códigos...

Ese «software», esencial para que funcione el ordenador, es el objeto de los «programadores» que lo crean o lo usan. Así, una parte del «software», o SW, es el llamado «sistema operativo», que preparan casas de SW

especializadas, y que sirve para unir entre sí los elementos de la máquina, del HW (además de otras funciones). El programador se encuentra, pues, con el Sistema Operativo ya listo; él no lo prepara; sólo lo usa.

Sistemas Operativos hay muchos, según las máquinas y las épocas. Baste saber que los «micros», o pequeños ordenadores, usan varios de ellos. Algunos comunes a varias máquinas, como el MS-DOS, el CP/M; otros específicos para un fabricante.

Sobre el sistema operativo, esto es, «soportado» por éste, se «montan» en el ordenador los lenguajes, las aplicaciones, etc. Así, el APL puede estar «soportado» por el PC-DOS (en el IBM PC, por ejemplo). Ello quiere decir que, al poner en marcha el ordenador, se suele «instalar», o «cargar» el sistema operativo en primer lugar, y luego el intérprete APL, «soportado» por aquél.

EL AREA DE TRABAJO

Y luego, ¿qué? Entonces, toma el timón el APL, y lo primero que hace es reservar una parte de la memoria del ordenador para que el usuario trabaje; ponga sus variables y programas, haga sus calculos, etc. Esa porción de memoria se denomina el *working storage*, o *área de trabajo*: WS. En el WS, de tamaño que depende de la máquina y del intérprete usados, pero siempre de un buen número de «bytes», o huecos para meter nuestros caracteres y números, se hará casi todo el trabajo del programador (más adelante veremos que se pueden trabajar con discos y otras «memorias externas»).

¿QUE ME QUEDA?

Cuando empieza a usar su WS, encuentra el operador un mensaje en la pantalla:

CLEAR WS

(WS "limpio"). Esto es: no hay nada, lo tiene todo para él solo. Si quiere saber qué tamaño tiene, puede escribir:

DWA

(se lee «quad WA»: WA es abreviatura de «working storage available» o «WS disponible»).

(el ordenador responde con el número de bytes disponible, que depende de máquina e intérprete).

Imaginemos que sea crea una variable:

CADA * 4 5 p 5

Si ahora preguntamos:
(¿qué variables hay?)

!VARS

(indica que en el WS está CADA).

CADA

¿QUE HE GASTADO YA?

Si queremos saber la memoria disponible, WA, comprobaremos que ha disminuido por el espacio que ocupa la tabla creada y su nombre.

En algún momento vimos que había «operadores bigamos» (que tenían dos argumentos, a su derecha y a su izquierda) y otros «monógamos» (que sólo los tenían a su derecha).

LLEGA «LA MONADICA»

Veamos ahora que casi todos los operadores pueden funcionar en ambas modalidades (más técnicamente llamadas *diádica* y *monádica*).

Operador	F. monádica	F. diádica
SUMA	+ 3 +3	1 3 + -2
RESTA	- 3 -2	1 4 - 5
MULTIPLICACION	* 1 *6	-1 3 * 5
DIVISION	/ 0.2 /5	15 1 5 + -2
POTENCIACION	** 2.718281 **1	-2.5 2 * 4
(SASTRE DEL AFL)	p 3 p 3 2 1	16 2 2 p 2 4 1
		2 4 1 2

De todos éstos, ya conocíamos las funciones diádicas, y la monádica del p (el «sastre» que «mide»). Pero ¿qué hacen los operadores anteriores en forma monádica?

— El + da como resultado el valor de su argumento con su mismo signo. Esto es, si el argumento es una variable:

A + 3 -2 5

(un vector)

y reclamamos:

+ A
3 -2 5

(resulta el valor de A).

Parece que no tiene mayor utilidad, pues lo mismo se consigue si se escribe simplemente:

A
3 -2 5

Con una diferencia: el + monádico permite asignar y en la misma instrucción mostrar el valor asignado:

```

+A + 3 -2 5
3 -2 5

```

Lo cual es útil a veces en programación, cuando se quiere ir sabiendo el valor que toman las variables que se crean.

— El - monádico cambia el signo a su argumento:

```

-A + 3 -2 5
-3 2 -5

```

UN X «SIGNIFICATIVO»

— El x monádico hace una cosa algo desconcertante:

resulta en 1 si el argumento es positivo, y en -1 si es negativo. Es muy útil, como luego veremos:

```

xA + 3 -2 5
1 -1 1

```

— El + aplicado en forma monádica obtiene el inverso del argumento:

```

+ 1 2 3
1 0.5 0.333

```

ERRORES EN EL DOMINIO

Qué pasa si lo aplicamos a 0 (operación que evidentemente no se puede realizar):

```

+ 0
DOMAIN ERROR

```

(error de «dominio»; esto es, la división por 0 entra dentro del «dominio» de las operaciones «prohibidas»)

— Por fin, el uso monádico de la potenciación, *, no es útil para el usuario no matemático; baste saber que es equivalente a elevar el llamado «numero e» al argumento. Por eso:

```

* 1
2.72

```

que es precisamente el valor del número e.

RESUMEN

Resumiendo el capítulo 3, recordaremos:

— El operador p en forma monádica da siempre como resultado un vector, lo cual proporciona sorpresas cuando se aplica en forma múltiple a un argumento:

```

A + 3 2 -1
pA ppp
3 1 1

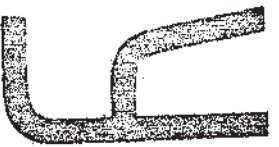
A + 6
pA ppp
Vect nulo 0 1

```

CAPITULO 4

- el «hardware» es la parte «física» del ordenador y el «software» son los programas, sistemas operativos, intérpretes y compiladores, etcétera;
- el «software» de programación y aplicación va generalmente «soportado» por uno de los «sistemas operativos» de la máquina;
- el APL proporciona al usuario un «área de trabajo», WS, donde colocar variables y programas, hacer cálculos, etc.;
- WA nos dice el WS disponible, no usado; VARS, las variables creadas;
- los operadores pueden funcionar la mayoría de las veces en forma monádica (un argumento, a su derecha), o diádica (dos argumentos);
- una operación no válida dará un mensaje de error: DOMAIN ERROR

A la «conversión» por el milagro: El bonito caso de los intereses al capital invertido, y otros problemas del que maneja dinero.—Porcentajes.—Beneficios y reparto de ganancias en el mus.—Promedios.



A parece hora de que veamos algún uso práctico de lo aprendido. Porque, con los tres capítulos anteriores y nuestros flamantes seis operadores, ya se pueden hacer bastantes cosas.

USOS PRACTICOS

Por ejemplo: obtengamos los beneficios (?) que proporciona un dinero puesto a interés compuesto. Conocemos la vieja fórmula:

$$C_n = C_0 (1 + r)^n$$

INTERES COMPUESTO

en que C_0 es el capital inicial, puesto al interés anual r por uno durante n años, y que se convierte en el capital C_n .

En APL:

```
C+1000
R+0.08
N+10
CN ← C × (1 + R) * N
```

CN
2158.92

Esto, con una buena calculadora, también se habría resuelto. Pero ¿y si queremos comparar los resultados con diversos intereses? Nada más fácil:

$$R+0.08 \quad 0.09 \quad 0.10 \quad 0.11$$

Y aplicamos la misma fórmula:

$$CN + C \times (1+R)^N$$

CN
2158.92 2367.36 2593.74 2839.42

No está mal, ¿verdad? También podemos probar con diversos números de años:

$$N+ B \quad 10 \quad 12 \quad 14$$

$$+CN + C \times (1+R)^N$$

1850.93 2367.36 3138.43 4310.44

Examinando este resultado, las cifras parecen algo raras. Y es claro, hemos usado R con cuatro valores y también cuatro valores de N. El ordenador ha calculado CN para pares de valores de R y N:

$$0.08, B \quad ; \quad 0.09, \quad 10 \quad ; \quad 0.10, \quad 12$$

que no es seguramente lo que queríamos (capital con interés fijo y años variables). Deberíamos primero dejar fijo R:

(por ejemplo)

$$R+0.10$$

$$+ CN + C \times (1+R)^N$$

2143.59 2593.74 3138.43 3797.5

Ahora si parecen lógicos los resultados. Ya veremos luego cómo se construye una tabla para diversos valores de R y de N. Por el momento lo podríamos hacer, en forma algo «chapucera», preparando una tabla de valores de R (con tantas columnas como años queramos, y tantas filas como necesitemos). Así:

$$R+3 \quad 4p \quad (4p0.08), \quad (4p0.09), \quad (4p0.10)$$

y otra tabla con los valores de N que queremos como exponentes:

$$N+ 3 \quad 4p \quad B \quad 10 \quad 12 \quad 14$$

Así:

R	0.08	0.09	0.10	0.08	0.09	0.10	0.08	0.09	0.10
N	0.08	0.09	0.1	0.08	0.09	0.1	0.08	0.09	0.1
B	10	12	14	10	12	14	10	12	14
8	10	12	14	10	12	14	10	12	14

Ahora si podemos aplicar la fórmula, y el resultado será una tabla.

	+ CN+CX(1+R)^N			
N:	8	10	12	14
R:0.08	1850.93	2158.92	2518.17	2937.19
R:0.09	1992.56	2367.36	2812.66	3341.73
R:0.10	2143.59	2593.74	3138.43	3797.5

(hemos despreciado los decimales).

Una tabla muy útil para toma de decisiones, por cierto.

VENDER Y GANAR

Veamos otro ejemplo. Tenemos una serie de artículos, 4, de los que vendemos ciertas cantidades:

ARTICULOS + 50 40 30 10

y los precios de cada artículo son:

PRECIOS + 200 500 300 100

Las VENTAS por cada artículo serán:

+ VENTAS + ARTICULOS * PRECIOS
10000 20000 9000 1000

Podemos calcular el total de ventas usando un nuevo operador: /, que significa «extender el operador de la izquierda al argumento de la derecha».

Así:

40000 +/VENTAS

(ha sumado todos los elementos en VENTAS). Si queremos saber, es un decir, la comisión sobre estas ventas, a razón del 10%, por ejemplo:

4000 + COMISION + 0.10 * +/VENTAS

PORCENTAJES

El cálculo de porcentajes es simple en APL: basta con teclear:

TASAS-10:20 30
+POR 100-(TASAS*+/VENTAS) + 100,

o también

+POR 100-0.01*TASAS *+/VENTAS

Podemos, igualmente, calcular el total de GASTOS de ventas, conociendo los elementos individuales:

20500 + GASTOS + 4000 + 5000 + 7500 + COMISION

Así, los beneficios serían:

NETO + (+/VENTAS) - +/GASTOS

(el segundo elemento no necesita paréntesis. ¿Por qué?)

Si sólo queremos saber si hubo ganancia o pérdida:

1 * NETO

(¡hubo ganancias! (NETO es positivo!).

EL MUS

Aún otro caso: una partida de mus, en la cual los jugadores han hecho cierta cantidad de puntos:

PUNTOS + 50 60 30 80

y deben repartirse (entre los 4) la «bacca» de 5.000 pesetas.

Para hacerlo en forma proporcional a los puntos:

BACA+5000 + REPARTO * PUNTOS * BACA + +/PUNTOS
 1136.36 1363.64 681.818 1818.18

PROMEDIOS

El promedio se calcularía:

$$1250 + \text{PROMEDIO} + (+/\text{REPARTO}) * \text{PUNTOS}$$

¡Lógicamente! Claro que habría sido más fácil:

$$1250 + \text{PROMEDIO} + \text{BACA} + 4$$

Pero eso no sería tan espectacular, digo yo.

En realidad, el cálculo de promedios es muy útil en la forma indicada cuando la masa de elementos es grande:

$$\begin{aligned} &A + 3 \cdot 2 \cdot 5 \cdot 4 \cdot 2 \cdot 6 \cdot B \cdot 3 \cdot 5 \\ &\text{PROMEDIO} + (+/A) + \rho A \\ &\text{PROMEDIO} \\ &3.77778 \end{aligned}$$

con lo cual no hemos tenido que contar físicamente el número de elementos en A: de eso se ha encargado monádicamente nuestro «sastre».

RESUMEN

En resumen:

— para calcular el interés compuesto, por ejemplo, no es preciso que las variables sean escalares; se pueden usar vectores y matrices, o tablas, con las debidas precauciones; esto es, sabiendo siempre qué estamos haciendo;

— en general, es útil aplicar fórmulas en las que los elementos sean vectores; el APL no tiene inconveniente en hacer operaciones con todos aquellos elementos al mismo tiempo;

— el operador / extiende la operación a su izquierda, al argumento a la derecha:

$$\begin{aligned} &+ / 3 \ 5 \ 6 \ 4 \\ &18 \\ &0 \quad \times / 2 \ 2 \ 3 \ 0 \end{aligned}$$

Los operadores - y ÷ producen resultados peculiares. Así:

$$\begin{aligned} &2 \\ &- / 3 \ 2 \ 5 \ 4 \end{aligned}$$

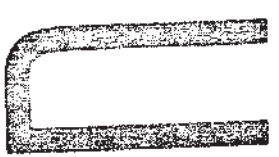
¿Por qué? (lo dejamos como ejercicio al lector).

CAPITULO 5

Otra letra griega.—El tren de los números naturales.—El operador de la suerte: ¿no es para preguntárselo dos veces?—Examinamos las operaciones a la tribu completa.—Una solución a la lotería por el APL.—La prueba del teléfono.—Y las letras, ¿quién las maneja?

UNA GRIEGA QUE TRAE COLA

Un nuevo e interesante operador se representa por la letra griega jota ι . Examinaremos en este capítulo su función monádica:



ι 5
1 2 3 4 5

Es decir: «crea» los números enteros desde el 1 hasta el argumento. Como es lógico:

ι -3
 ι 3.5

no tienen sentido. Ni tampoco:

ι 3 5 2

Esto es, ι se aplica sólo a un escalar entero y positivo. Y, en cambio, su resultado es siempre un vector: así:

ι 1 ι 1
1 2 1 1
 ι 1

Y por otra parte:

... 0 ...
(vector nulo),

que es otra forma de «crear» a nuestros amigos los vectores nulos. De futura e importante aplicación, aseguro al lector.

SERIES CURIOSAS

Una simpática forma de generar una serie de números con 1 es ésta:

40 + 1 5
41 42 43 44 45

Los números del 1.001 al 2.000 se obtendrán de:

NUM + 1000 + 1 1000

Y los números pares del 1 al 200:

PARES + 2 x 1100

De igual forma, los 50 primeros números impares:

IMPARES + -1 + 2 x 150

OPERADOR PREGUNTON

Hoy vamos a examinar también un curioso operador que va a hacer las delicias de quienes odian la precisión y el destino. Este operador, ? , esto es, un simple signo de interrogación, se lee **aleatorio**, esto es, selección al azar de un conjunto. Tiene el operador ? usos en varias áreas, como vamos a ver, y su potencia es considerable. Puede funcionar tanto en forma monádica como diádica. Así:

A+27
? A
4
? A
21

Esto es, cada «aplicación» monádica de ? a un escalar resulta en un número al azar de su entorno numérico.

¿Y si se aplica a un vector?:

? 20 15 10
10 8 3
? 20 15 10
1 11 7

Esto es: resulta otro vector de números al azar.

Así, si queremos 10 números al azar entre el 1 y el 100:

? 10 p 100
94 39 52 84 4 6 53 68 1 39

LA «LOTO»

Esto puede ser útil, por ejemplo, para «extraer» los 5 números de la lotería primitiva:

2 5 9 30
4 21 35 30 47

¡O incluso todos los números de la Lotería Nacional!

EL PROBLEMA DE LOS TELEFONOS AL AZAR

En realidad, hubo en su día una aplicación práctica que dejó no pocos convencidos de la eficacia del APL. Se quería obtener una serie de 1.000 números comprendidos entre el 50.000 y el 70.000 para hacer ciertas pruebas en una Compañía Telefónica. Los lenguajes convencionales de programación habían hecho una labor compleja y muy lenta. En cambio, vease la elegante solución APL:

```
499999 + ? 1000 # 20001
```

¿Por qué 499999? ¡Ah! Hay que pensarlo un poquito.

PREGUNTAS BIGAMAS

El uso diádico del operador «aleatorio» ? es interesante:

```
5 ? 10  
9 6 3 8 7
```

El resultado, como se ve, es un **vector** que tiene los elementos indicados por el argumento izquierdo, elegidos en forma aleatoria entre la serie natural que indica el argumento derecho y **sin repetirlos**.

Por eso:

```
20 2 10  
DOMAIN ERROR  
20?10  
^
```

no se puede ejecutar y

```
10 ? 10  
8 10 9 5 4 3 1 6 2 7
```

dará **todos** los números del 1 al 10, pero en **distribución al azar**. Obsérvese que si queremos 10 números aleatorios, todos entre el 1 y el 10, no se debe usar la fórmula anterior, que no daría números **todos** entre el 1 y el 10 en selección paritaria, sino

```
? 10 # 10  
4 3 10 8 8 7 1 7 9 3
```

Véase que se pueden (y en general ocurrirá así) repetir algunos números en la serie del 1 al 10.

SOPA DE LETRAS

Pasemos a otro tema. Hasta ahora hemos tratado sólo con «números». Caramba, parece que me he olvidado de las «letras»! Estoy dando la impresión —completamente errónea, por cierto— de que el APL sólo sirve para las matemáticas. Eso, en un hombre que es poeta (además de ingeniero) es imperdonable.

Bien: vamos a remediarlo.

Así es: podemos tratar con letras, con caracteres, igual que con números. No podremos «operar» con ellos, claro, pero sí hacer muchas otras manipulaciones.

Empecemos por aclarar que los caracteres, o letras, o signos —llámenseles como se quiera—, deben escribirse en APL **entre comillas**, para di-

Referenciarlos de las variables. Así, la letra A se puede asignar a una «celda», a una variable:

```
LETRA = 'A'
LETRA
A
```

Si «medimos» LETRA:

```
p LETRA
```

¡Eso es! Lo que usted ha adivinado: una letra sola es un escalar. Ahora:

```
LETRAS = 'PEPE'
p LETRAS
```

4

¡De nuevo acertó! Un conjunto de letras, una palabra, es un vector:

```
p FRASE + 'ESTO ES APL'
```

11

LOS BLANCOS CUENTAN

Esto deja claro que los «blancos» también cuentan, son caracteres ellos mismos. Y si no, pregúntele a un cajista de imprenta si es verdad que coloca un «tipo» especial (sin letra marcada) entre palabras, para que se «imprima» un blanco, o si lo puede dejar sin nada entre aquéllas. Para él, el «blanco» es bien real.

Podemos, igualmente, crear matrices de letras:

```
TABLA + 4 4 p 'LUISPEPEANA JOSE'
```

Obsérvese que hemos debido dejar un blanco, para que resulte:

```
TABLA
LUIS
PEPE
ANA
JOSE
```

Podemos aplicar el operador p como con números:

```
7 p FRASE
ESTO ES
4 p FRASE
ESTO
```

O equivocarnos:

```
3 4 p 'LUISPEPEJORGE'
LUIS
PEPE
JORGE
```

¿MATRICES VACIAS?

Puede parecer extraño, pero podemos crear una matriz de blancos:

```
M + 4 4 p ' '
```

En M habrá, en efecto, una tabla 4x4 hecha de blancos; no vacía, aclaramos. Y una matriz muy útil, porque es como un molde que se puede rellenar de lo que convenga más tarde.

Aún hay más: puede ser necesario, y así lo mostraremos, crear una tabla, por ejemplo, de nombres, con una fila inicial, a la que se añadirían otras en el futuro:

```
NOMBRES + 1 10+ JOSE LUIS
```

e incluso, comenzar con una tabla de 10 columnas pero ninguna fila:

```
NOMBRES + 0 10 #
```

NO SE VE PERO SE TOCA

¡Esta sí que es una «tabla fantasma»! Tiene 10 columnas, pero ninguna fila. ¿Puede ser? Pues, sí puede ser y es. Aunque parezca una tabla bien poco consistente... En realidad

```
# NOMBRES
```

```
0 10
```

no deja lugar a dudas. Ya sabemos que nuestro «sastre» p es un operador serio, que no nos engañaría si quisiéramos aplicarlo a algo no existente. Por supuesto, al teclear

```
NOMBRES
```

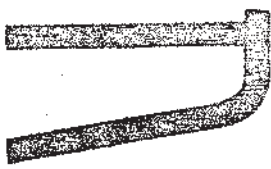
el ordenador «salta» una línea, y no muestra nada. Pero a eso ya estamos acostumbrados a estas alturas.

RESUMEN

- El operador `!` (iota) genera serie de números naturales... y también el vector nulo.
- El operador `?` (aleatorio), en forma monádica, suministra números al azar, que pueden repetirse si se obtienen de un mismo argumento; en cambio, en forma diádica genera números aleatorios no repetidos.
- Se puede «crear» y trabajar con escalares, vectores y matrices de caracteres; hay que escribirlos entre comillas.

CAPITULO 6

La iota bigama y la h perdida en el nomulo.---Tablas de nombres.---Una coma mágica: pega variables o las aplasta.---Haciendo vectores de los escalares.



LA IOTA BIGAMA

LESTRO operador i también funciona en forma diádica, así:

$$2 \quad 4 \quad 3 \quad 2 \quad 5 \quad 1 \quad 3$$

En este caso, le llamamos «índice», porque busca el lugar que ocupa el argumento de la derecha en el vector argumento izquierdo

$$2 \quad 3 \quad 4 \quad 2 \quad 4 \quad 1 \quad 4$$

Ello indica que busca la primera ocurrencia del argumento derecho. Con letras:

$$3 \quad \text{'CARLOS' } \wedge \text{'R'}$$

El argumento derecho puede ser escalar, vector o matriz:

$$3 \quad 5 \quad 6 \quad 2 \quad \text{'CARLOS' } \wedge \text{'ROSA'}$$

Observemos un resultado peculiar:

4 3 2 5 1 10
5

Esto es, si el argumento derecho no se encuentra en el izquierdo de búsqueda, el resultado es una unidad más que la longitud de ese vector-argumento:

'CARLOS' 1 'PEPA'
7 7 7 2

La iota diádica puede servir para «localizar» elementos dentro de un argumento:

FRASE ← BUSCAMOS UNA H EN LA FRASE'
FRASE 1 'H'
14

Fijense que cuentan los blancos, por supuesto. O bien:

A + 4 6 3 10 5 2

Busquemos donde está el número 10:

A 1 10
4

LA COMA «CELESTINA»

Hemos dedicado un cierto espacio en el capítulo 5 a las tablas de nombres, entre otras razones porque tienen muchas aplicaciones. Ahora vamos

a examinarlas más de cerca, usando un nuevo operador, que se escribe simplemente, (coma), y que leemos «catenación», o unión en cadena.

El operador «catenación» permite unir elementos:

4 5 4 , 5

Dos escalares se han unido para formar un vector.

Igualmente, con letras:

'AB', 'CD'
ABCD

O uno dos vectores:

A+3 2 4
B+5 0 2 6
A, B
3 2 4 5 0 2 6

Y si son de caracteres:

NOMBRE ← 'LUIS DE '
APELLIDO ← 'CAMDENS'
NOMBRE, APELLIDO
LUIS DE CAMDENS

Observemos que hemos dejado un blanco al final del nombre para que no quedase «pegado» el apellido al catenar.

Pondríamos así:

NOMBRES ← 4 5 p 'LUISAPEPE JUAN JORGE'
NOMBRES

LUISA
PEPE
JUAN
JORGE

y también los apellidos:

```
APPELLIDOS+4 6º PEREZ ABAD GLEZ GARCIA
APPELLIDOS
PEREZ
ABAD
GLEZ
GARCIA
```

Ahora podemos catenar las dos matrices:

```
NOMBRES, ' ', APPELLIDOS
LUISA PEREZ
PEPE ABAD
JUAN GLEZ
JORGE GARCIA
```

Es interesante ver que, así como dos matrices debían (lógicamente) tener el mismo número de filas para su catenación, se puede intercalar en una operación de catenación un escalar, en este caso el blanco.

UNA BODA DESIGUAL

Si queremos «catenar» en el otro «sentido», esto es, las filas sobre las filas, deberemos tener en ambas matrices el mismo número de columnas. Por eso:

```
NOMBRES, [1] APPELLIDOS
```

Se coloca [1] para indicar que la catenación se debe ejecutar por la primera dimensión, esto es, por las filas. Si se omite, el intérprete APL «entiende» [2], o sea, por la segunda dimensión, por las columnas (que es lo que arriba hizo). Por eso, se podría haber escrito antes:

```
NOMBRES, [2] ' ', [2] APPELLIDOS
```

Pero al catenar ahora por las filas, [1], el ordenador rehusa ejecutarlo, obviamente no «casan» 5 columnas con 6 columnas, y responde con un mensaje de error:

```
NOMBRES, [1] APPELLIDOS
LENGTH ERROR
NOMBRES, [1] APPELLIDOS
^
```

Claro que si las dimensiones de NOMBRES, fuesen 4 6 y no 4 5 el «casamiento» de ambas matrices no habría tenido problema. Veamos ahora para qué sirve crear matrices vacías:

```
LISTA ← 0 10 ' '
```

Queremos una lista de artículos de almacén, que tendremos en la variable LISTA. La creamos de 10 columnas, que parece adecuado, pero sin ningún elemento. Cada vez que añadamos un artículo, ejecutaremos:

```
LISTA+LISTA, [1] 'MARTILLOS '
LISTA+LISTA, [1] 'ALICATES ' ,
```

etcétera, con lo cual habremos catenado por las filas a la antigua matriz LISTA un vector de 10 caracteres con el nombre del nuevo artículo, y la matriz resultante con una fila adicional, se asignará el antiguo nombre LISTA. La vieja queda sustituida por la nueva de esta forma. ¿Sirven para algo las matrices vacías o no?

¡VIVA LA ECONOMIA!

Así, uno evita crear, como se hace en otros lenguajes, la matriz con el número máximo de líneas que se estima necesitar, e ir llenándola poco a poco: eso gasta mucha memoria y entre otras cosas, parece una «chapuza». Creo más elegante ir añadiendo filas cuando se necesitan, «dinámicamente».

Hay que tener cuidado al añadir elementos a la matriz, que no tengan distintas dimensiones; por ejemplo:

```
CAJA + S 6 p 1 30
```

crea una tabla de números del 1 al 30, en 5 filas.

Si queremos añadir más números:

```
CAJA + CAJA, [1]31 32 33 34 35 36
```

Si hubiéramos escrito:

```
CAJA + CAJA, [1]31 32 33 34
LENGTH ERROR
```

¡El mensaje de error nos lo hemos merecido!

Un último ejemplo:

```
+ A + 2 6 p 10 + 1 12
11 12 13 14 15 16
17 18 19 20 21 22
+ B + 3 6 p 18 ? 100
44 77 49 27 31 40
22 53 91 92 16 10
57 58 42 99 14 6
A, [1]B
11 12 13 14 15 16
17 18 19 20 21 22
44 77 49 27 31 40
22 53 91 92 16 10
57 58 42 99 14 6
```

¿Alguna duda?

LA COMA PILON

Ya hemos visto nuestra coma diádica, «casamentera»; veamos ahora su uso monádico que paradójicamente hace casi todo lo contrario: aplasta, reduce a un finísimo vector a sus argumentos; ¡contra la coma monádica no «valen argumentos»! Así:

```
CAJA + 3 3 p 1 9
+ HILD + , CAJA
1 2 3 4 5 6 7 8 9
```

Y esto lo hace con vectores, con matrices y con hipermatrices:

```
CAJON + 2 3 4 p ABCDE
HILITO + , CAJON
HILITO
ABCDEABCDEABCDEABCD
```

LA COMA SOCIALISTA

Incluso a los escalares los convierte en vectores con lo cual, en realidad, les eleva el «status»:

```
N + 5
,N
p,N
1
```

Así, podemos decir que la coma monádica es ¡la gran igualadora social!

RESUMEN

En este importante capítulo hemos visto:

— el uso diádico de la $+$, como «índice», para encontrar la situación de los elementos del argumento derecho en el vector argumento izquierdo;

CAPITULO 7

— los usos de la «coma»:

- a) en forma monádica, como «igualador» social, esto es, para convertir en vector los argumentos;
- b) en forma diádica, como «cola universal» que une argumentos a lo largo de la dimensión que se especifica entre corchetes: [1], [2], etc.

La delta al revés y el APL servicial.—Corregir sobre la marcha.—La cesta de gummies.—Haciendo sencilla a Su Majestad la Estadística.—¿Qué pasa si se va la luz?



A tenemos bastantes elementos, aunque por supuesto, no todos, ni siquiera la mayoría, para hacer «programas». Esto quiere decir lo siguiente: para resolver un problema podríamos ir dando las órdenes, o instrucciones, tecleándolas directamente; el intérprete APL se encargara de ejecutarlas una tras otra, e irnos dando resultados parciales que usaríamos en la siguiente etapa. Pero es conveniente «reunir» todo un bloque de instrucciones, si ese problema se va a presentar a menudo, en una «caja», para usar la «caja» cada vez, en lugar de reinventar en cada ocasión la rueda!

EMPIEZA LA FUNCION

Esas «cajas» se llaman en APL, **funciones**; se les da un nombre conveniente (tiene que empezar por una letra, como las variables), que no coincide, claro, con el de una variable ya creada. Y se tecldea así:

```
[1]  ▽ FUNCION N
```

El APL, al «leer» ▽ entiendo que lo que sigue es el nombre de una función, o programa; lo almacena con ese nombre, escribe [1], esto es, número de la primera línea, y espera a que el programador teclee la primera instrucción:

```
[1]  *PROGRAMA PARA ELEVAR AL CUADRADO*  
[2]
```

Cada vez que el programador añade una línea (en este caso, un simple mensaje, un vector de caracteres), el ordenador añade un nuevo número de línea para ayudar a su «amo»: ¡buen chico!

Por cierto, junto al nombre de la función hemos escrito N; el ordenador ha «entendido» que en la función se usará una variable con el nombre N y que su valor es el que hayamos puesto junto al nombre de la función, o definido antes así:

```

*FUNCION N
[11] *PROGRAMA PARA ELEVAR AL CUADRADO*
[21] N * 2
[31] V
```

LA DELTA QUE ABRE Y CIERRA EL TELON

Como ya hemos terminado, volvemos a poner el símbolo V (que se le da) y que indica al intérprete que hemos terminado de escribir la función; ahora sí decimos:

```

*FUNCION 7
PROGRAMA PARA ELEVAR AL CUADRADO
49
```

El ordenador ha reconocido que invocamos una función que tiene ya definida, con el argumento, o parámetro, 7; ejecuta lo que diga cada línea, y se detiene en V.

Línea 1: muestra un vector de caracteres

Línea 2: eleva al cuadrado el parámetro y lo muestra.

Teclamos ahora:

```

*FUNCION 2 3 -5
PROGRAMA PARA ELEVAR AL CUADRADO
4 9 25
```

Esto es, el parámetro N ahora es un vector; como bien sabemos, el vector se eleva al cuadrado elemento a elemento.

A mí, al menos, empieza a molestarne ver el mensaje

```
PROGRAMA PARA ELEVAR AL CUADRADO
```

cada vez que ejecutamos la función; vamos a cambiarlo de forma que sea un comentario sobre lo que hace el programa, útil para quien desee saber lo que no sea el autor (aunque a veces a éste se le olvida para qué ha escrito en su momento).

«Abramos» la función; esto es, mostremos su contenido. Esto se hace así:

```
*FUNCION [01]
```

(al colocar entre corchetes el *quad.* el APL entiende que debe mostrarse la función completa):

```

*FUNCION [01]
[01] *FUNCION N
[11] *PROGRAMA PARA ELEVAR AL CUADRADO*
[21] N*2
[31]
```

La función queda «abierta», esperando que añadamos cosas.

En su lugar, escribimos:

```

[31] [1] N PROGRAMA PARA ELEVAR AL CUADRADO
[21] V
```

Escribimos [1] para modificar el contenido de la línea 1; [N] indica que lo que sigue es un comentario no ejecutable; el ordenador escribe ahora [2]; como no queremos modificar la línea 2, teclamos V y la función queda

cerrada, con la línea 1 modificada para guardar un comentario que no se ejecutará:

```

1 4
9 16
FUNCTION 2 2 2 4

```

Si queremos mostrar la función, pero no modificarla, escribiremos:

```

*FUNCIONCUBO
[0] FUNCION N
[1] A PROGRAMA PARA ELEVAR AL CUADRADO
[2] N*2

```

Abrir y cerrar funciones para arreglar errores, añadir o quitar es esencial para el programador; observemos:

```

*FUNCION [20]
[2] N*2
[2] [1.1] EL CUADRADO DE

```

(Intercala la línea 1.1 entre la 1 y la 2)

```

[1.2] N
[1.3] 'ES'
FUNCTION 2 4
EL CUADRADO DE
2 4
ES
4 16

```

Si ahora mostramos la función completa:

```

*FUNCIONCUBO
[0] FUNCION N

```

```

[1] A PROGRAMA PARA ELEVAR AL CUADRADO
[2] 'EL CUADRADO DE'
[3] N
[4] 'ES'
[5] N*2
[6] CA110

```

Hemos aprovechado para eliminar con A1 la línea 1 (no esencial) y luego cerrar la función.

FUNCIONES QUE «RESULTAN»

Vamos a ver ahora una modificación a la «sintaxis» de las funciones, que nos permitirá convertirlas en **explícitas**, esto es, usar el resultado que produzcan como argumento para otra función. Esto se consigue modificando la línea 0:

```

*FUNCION [00]
[0] FUNCION N
[0] Z + FUNCION N
[1] [5] Z + N * 2

```

Esto es: el enunciado, la línea 0, de la función, se asigna a una variable que inventamos, Z por ejemplo; y a esa **misma Z** se le asigna **dentro** de la función la variable que queremos como resultado de la función (en este caso, el cuadrado calculado). Veamos para qué sirve:

```

* Z + SUMA N
[1] Z + +/N

```

Ejemplo:

```

18 SUMA 5 6 7

```

```
07 * FILA N
[1] 2 4 1 N V
```

```
1 2 3 4
      FILA 4
```

Y ahora ordenamos:

```
FUNCION: SUMA FILA 5
EL CUADRADO DE
15
ES
225
225
```

¿que ha ocurrido? El resultado de:

```
1 2 3 4 5
      FILA 5
```

ha quedado como argumento de:

```
15
      SUMA 1 2 3 4 5
```

Y este resultado ha servido como argumento de:

```
15
      FUNCION 15
EL CUADRADO DE
```

```
ES
225
225
```

Esto es:

```
FUNCION (+) SUMA (+) FILA
```

Eso no habria ocurrido de no ser las funciones **explicitas**, esto es, si en las lineas 0 y otra no hubieramos asignado a Z los resultados. Estas funciones se llaman **implicitas**, como:

```
07 FILA N
[1] 1 N V
```

Aqui el «resultado» de ejecutar:

```
1 2 3
      FILA 3
```

es siempre un vector 1 2 3, pero no es «escupido» por la función como su resultado, su valor, sino mostrado en pantalla (o impresora). Por eso:

```
1 2 3
      SUMA FILA 3
VALUE ERROR
```

El APL ejecuta la primera orden, FILA 3, pero al intentar encontrar el argumento de SUMA, no lo halla porque FILA no tiene un valor, por ser implícita y no explícita. ¿Un poco más claro?

Estas funciones implícitas son muy útiles, sobre todo cuando se usan dentro a su vez de otras funciones, para ir calculando pequeños trozos del problema.

BAH, LA ESTADISTICAL

Veamos un ejemplo dentro del área de la Estadística. Con los elementos que forman conjuntos de números, es habitual el cálculo de su varianza, que se puede definir como suma de los cuadrados de las desviaciones del conjunto, o vector de elementos, respecto de su media aritmética.

Así supongamos el conjunto:

SERIE + 3 6 0 -2 4 3 5

La media aritmética se calcularía así:

```

^ X + MEDIA N
[11] X ← (+/N) + /N^
  
```

Esto es, la suma de los elementos del argumento N dividida por el número de elementos.

Podemos también crear una función para calcular la suma:

```

^ X + SUMA N
[11] X ← +/N^
  
```

con lo cual reescribiremos la media:

```

^ X + MEDIA N
[11] X ← (SUMA N) / N^
  
```

(aunque es rizar el rizo). Escribamos de nuevo la función que calcula el cuadrado, simplificándola:

```

^ FUNCION
[61] [A21]
  
```

```

[31] [A31]
[41] [A41]
[51] [E01]
[01] Z+FUNCION N
[11] *EL CUADRADO DE*
[51] Z+N*2
[61] [A11]
[21] ^
  
```

con lo cual hemos dejado sólo la línea del cálculo del cuadrado (siempre explícita).

Ahora hagamos una función para hallar la desviación:

```

^ X + DESVIACION N
[11] X ← N - MEDIA N^
  
```

que restará a cada elemento de N su media.

En fin; definamos la varianza:

```

^ Y + VARIANZA N
[11] Y ← MEDIA FUNCION DESVIACION N ^
  
```

Fijémosnos en que DESVIACION usa dentro a MEDIA, MEDIA usa a SUMA, VARIANZA usa MEDIA, CUADRADO y DESVIACION.

```

^ FUNCION [E01]
[01] Z+FUNCION N
[01] Z+CUADRADO N
[11] ^
  
```

(hemos cambiado FUNCION por CUADRADO para mayor claridad)

```

^ VARIANZA [E01]
[01] Y+VARIANZA N
[11] Y+MEDIA FUNCION DESVIACION N
[21] [11] Y + MEDIA CUADRADO DESVIACION N^
  
```

Podemos, pues, ejecutar:

```
VARIANZA SERIE
6.77551
```

Aún más, en Estadística se define la **desviación estándar** como raíz cuadrada de la varianza; definimos (aunque nos «pusemos») la función:

```
√ M + RAIZ N
C11 M + N * 0.5√
```

```
PK+ESTANDAR N
C11 X + RAIZ VARIANZA NV
```

Podremos ejecutar:

```
ESTANDAR SERIE
2.60298
```

LA CESTA DE CEREZAS

Para aquellas personas acostumbradas a tener **un solo programa** en su área de trabajo, puede parecer sorprendente el que podamos **crear muchas pequeñas funciones** para resolver problemas sencillos, y que cada una de ellas pueda invocarse directamente desde otra función, e incluso en varias funciones. Eso ahorra mucho esfuerzo de programación y simplifica la presentación de los programas. Es más, estoy convencido de que, en APL, programar funciones largas es un error anti-APL. Lo elegante y práctico es tener **funciones breves** (de no más de 10-15 líneas cada una; algunas de 1 o 2 líneas tan sólo). Se puede incluso tener una **pequeña biblioteca de funciones útiles**, que resuelven pequeños problemas, y usarlas en nuestros programas cuando se necesitan.

¿QUE TENEMOS?

Acabamos de aludir a «bibliotecas de funciones útiles», a «usar» funciones en una biblioteca para nuestro programa actual. ¿Cómo se hace eso? Ante todo, pensemos en nuestra «área de trabajo», nuestro WS. Dentro de él tenemos nuestras variables:

GVARS	HILITO	HILIT	LETRA	LETRAS	LISTA	N
GASTOS	MEI	PREZ	N	NETO	NOMBRE	NUMEROS
MENU	PRELIDOS	PROMEDIO	PUNTOS	R	REPARTI	SERIE
NOTAS	VERDAS					
TABLA						

y nuestras funciones, que se listan así:

FINES	CALCULO	CALENDAR	CARBAS	CHARRAS	DEVIACION	ELIMINAR
REPREGAR	FICHERO	FILEA	LISTAR	MEIDA	RAIZ	SUMA
ESTANDAR						
VARIANZA						

Cuando hemos acabado el trabajo, o lo suspendemos, o simplemente queremos prever de cuando en cuando que pueda «corrarse» la luz, y perder todo lo hecho, «guardamos nuestro WS» ¿en dónde? Habitualmente será en un diskette, o cassette, o disco duro; esto es, en una memoria externa. ¿Cómo se «guarda», o salva un WS? Teclando:

```
)SAVE TRABAJO
```

Así (la sintaxis de esta «orden» puede variar de un ordenador a otro), se coloca nuestro WS, que hemos «bautizado» como TRABAJO, en el diskette, etc. Y cuando lo queramos recuperar, ordenaremos a nuestro intérprete APL:

```
)LOAD TRABAJO
```

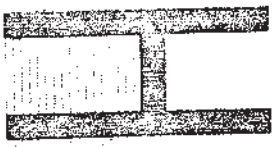
una vez colocado el diskette en su sitio adecuado. Otros modos de acceder a la memoria externa son también empleados, pero aquí no es preciso examinarlos.

RESUMEN

En este capítulo hemos aprendido:

- a crear «funciones», programas que reúnen varias instrucciones, y se puedan ejecutar conjuntamente;
- que las funciones deben ser breves y muchas, mejor que pocas y largas;
- que esas funciones se pueden usar en otras funciones y guardar en «bibliotecas» para su uso posterior, etc., y que para ello suele ser preciso crearlas como funciones explícitas, que «devuelven» un resultado usado por la otra función;
- que, para listar las funciones de nuestro WS, se usa el mandato)FNS;
- cómo guardar, salvar nuestro WS con el mandato)SAVE, y cómo traerlo, copiarlo a nuestra área de trabajo de nuevo (borrando todo lo que en él haya), con el mandato)LOAD.

Del canto al coro y del coro al caño.—Las propiedades singulares de la T, del derecho y del revés.—Barriendo la casa



ASTA este momento ha habido una temible barrera, una división absoluta en toda nuestra exposición entre **letras** y **números**. Pero el caso es que los números están compuestos de **cifras**, que como símbolos que son de cantidad son unas figuras; esto es, no hay ninguna razón para que no se puedan también ver **como letras**. Y es así como figuran dentro de los textos, como caracteres:

NACI EN 1950 Y TENGO 2 HERMANOS

En esta frase, '1950' y '2' son cinco caracteres, que podrían haber sido originados como números por un cálculo, y que podríamos querer «insertar» en la frase. ¿Cómo hacerlo? Si escribimos

*NACI EN '1950, '2 Y TENGO' 2, *HERMANOS*

DOMAIN ERROR

*NACI EN '1950, 'Y TENGO' 2, *HERMANOS*

El error indicará que no se pueden «mezclar peras con manzanas». Y aquí aparece otro operador, **F** (**formatear**), más o menos una T con un cerito en su palo mayor.

LA T MAGICA

El operador **F** convierte números en caracteres:

N+23
C+ * 23

asi:

P N

(vector nulo). Pero:

2 P C

(puesto que es un vector de 2 caracteres). El mismo operador puede hacer lo mismo con vectores o con matrices de números:

```
* V + 3 2 6 10
3 2 6 10
```

que es ahora un conjunto de 8 caracteres.

Por eso, podemos cómodamente mezclar letras y números:

```
'NACI EN ' , (#1950) , ' Y TENGO ' , (*2) , ' HERMANDOS'
NACI EN 1950 Y TENGO 2 HERMANDOS
```

Observe el lector los blancos que hay que dejar entre letras y cifras. Por cierto, al convertir tablas de números, decimales o no, conviene definir la separación física entre cifras. Eso se consigue colocando un argumento de 2 elementos a la izquierda de \bar{P} , así:

```
5 1 * 3 2.1 4 5
3.0 2.1 4.0 5.0
```

Esto es particularmente útil al formatear tablas:

```
4 2 * 4 4 P \ 16
DOMAIN ERROR
```

En efecto: 4 espacios y 2 decimales es menos de lo que ocuparía, por ejemplo, el número 16 formateado: 16.00 (5 espacios). Habrá que decir:

```
TABLA + 5 1 * 4 4 P \ 16
TABLA
1.0 2.0 3.0 4.0
5.0 6.0 7.0 8.0
9.0 10.0 11.0 12.0
13.0 14.0 15.0 16.0
```

Si queremos capturar en un vector una tabla:

```
'ESTO ES LA TABLA ' , TABLA
LENGTH ERROR
```

pues, en efecto, no tienen igual dimensión:

```
4 20
* TABLA
```

(4 líneas de 20 caracteres, o columnas).

En todo caso, se podría «aplanar» la tabla o «travélarla», con la coma monodimensional.

UNA T VERDUGO (PORQUE «EJECUTA»)

Por otro lado, si tenemos un dato «alfanumérico», esto es, letra o cifra pero «carácter» y queremos operar con el debemos «ejecutarlo» primero, o sea, convertirlo a número; esto lo hace el operador \bar{I} (formatear al revés):

```
DOMAIN ERROR
```

```
3 * B ← -3
```

(pues pretendemos multiplicar un número por un carácter). En cambio:

```
3 * B
```

El carácter 3 se ha convertido en el número 3 mediante el operador \leftarrow («ejecutar»). Así el operador \leftarrow permite convertir en «valor APL» lo que es carácter, si ello es posible. Por ejemplo:

```
A ← 3 2 5
```

A es una variable, claro. Si escribimos:

```
A
```

(pues «A» no es una variable, sino un carácter). En cambio, si «ejecutamos» «A», lo convertimos en variable:

```
A
```

Así:

```
DOMAIN ERROR
```

```
3 * 'A'
```

```
9 6 15
```

Esta es una de las posibilidades de mayor potencia en APL, imaginemos que tenemos esta frase:

```
FRASE ← 'A ES UNA VARIABLE'
```

Si conseguimos aislar la A, por ejemplo, así:

```
1 ↑ FRASE
```

```
A
```

(selecciona el primer carácter de FRASE; lo veremos mejor luego).

```
1 ↑ FRASE
```

```
3 2 5
```

¿Nos hemos convencido? Pues veremos muchos usos de este singular operador.

Por último quizá nos hayamos a estas alturas percatado de que \leftarrow y \leftarrow operan en sentido contrario, precisamente así:

```
3 * A ← A
```

```
9 6 15
```


LIMPIEZA GENERAL

Recordemos que en nuestro WS se guardan todas las variables y todas las funciones que hayamos creado y que se visualiza su lista con)VARS y)FNS. Ahora bien: después de un buen rato de trabajar y aunque el tamaño del WS es bastante grande (recuérdese que lo disponible se sabe con WA), puede ocurrir que nos «comamos» la memoria disponible, porque

estamos el WS de variables y aún de funciones no utilizadas.
¿Qué hacer? Pues lo que haríamos en una pizarra: borrar todo lo inútil. Para ello se usa el comando)ERASE, seguido de las variables y funciones no necesarias:

```
)VARS
GASTOS HILLO LISTA M
)FNS
ESTANDAR MEDIA RAIZ
)ERASE GASTOS RAIZ
)VARS
HILLO LISTA M
)FNS
ESTANDAR MEDIA
```

Con esta simple operación se deja más espacio disponible, como se puede comprobar con el mandato WA. En todo caso, mucho cuidado con que no se produzca el mensaje:

```
WS FULL
```

(cartera de trabajo llena).

RESUMEN

Resumiendo el capítulo:

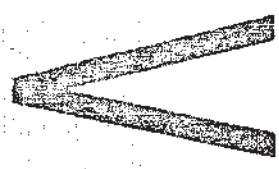
— **F** y **T** sirven para convertir en «objetos APL» o números las expresiones en caracteres o para convertir en caracteres los números;

— para hacer el «formateo» o conversión a caracteres de números, más preciso, y definir el número de cifras decimales, se usa un argumento izquierdo de **F**, que define el número de espacios y de cifras decimales que corresponden a cada número formateado;

— el mandato)ERASE borra del WS las variables y funciones innecesarias para evitar el mensaje WS FULL.

CAPITULO 9

Como colar a Pepe en la fila del cine.—Un paréntesis poco recordando.—Ordenando lo que está en desorden.—El primero y el último de la clase.—¿Quién se va de viaje?—Movándose por la función.—Compresión viene de comprimir.—Variables limpias



IMOS antes cómo se podía saber la posición de un elemento dentro de un vector, mediante el operador $\text{I}(\text{tota})$ en forma diádica:

$$3 \quad \text{'ABCD' } \wedge \text{'C'}$$

Veamos ahora la que podemos llamar operación inversa: extraer un elemento determinado de un conjunto. Así:

$$C \quad \text{'ABCD' } \text{I} \text{'3'}$$

o si queremos extraer varios:

$$CD \quad \text{'ABCD' } \text{I} \text{'3' } \text{'4'}$$

Esto nos puede servir también para sustituirlos por otros:

$$M + \text{'ABCD'}$$

$$M \text{'3' } \text{'4' } + \text{'XY'}$$

$$M$$

$$ABXY$$

Con números:

```

      8 + 110
      512 43 + 12 14
      9
      1 12 3 14 5 6 7 8 9 10

```

Si tenemos un vector de caracteres:

COLA ← 'LUISJUANANNA'

¡ALA COLA, ALA COLA!

Podemos cambiar parte del vector:

```

COLALA ← 'AJ'+PEPE?
COLA
LUISPEPEANNA

```

En forma de matriz:

```

      FILA+3 4+COLA
      FILA
      LUIS
      PEPE
      ANNA

```

MOSTRAR LA ROPA INTERIOR

Si queremos extraer elementos de la matriz:

```

      FILA1 3;1
      LUIS

```

```

ANNA
      FILA1:2]
      UEM
      FILA1:3]
      1

```

Si queremos extraer todas las filas:

```

      LUIS
      PEPE
      ANNA
      FILA1 2 3;1

```

Si deseamos todas las filas y columnas:

```

      LUIS
      PEPE
      ANNA
      FILA1;1

```

¡Fijémosnos que utilizamos corchetes, no paréntesis!

Todo esto está muy bien, pero supongo que el lector habrá echado en falta cómo poner en orden números y letras. Eso se consigue con el operador Δ (orden ascendente) y ∇ (orden descendente).

ARRIBA Y ABAJO

Pero, ¡atención! Estos operadores no ordenan por sí, sino que dan la posición que ocupan los elementos que deben estar en ese orden. Esto es

```

NOTAS ← 3 5 0 8 6
      ▲ NOTAS
      3 1 2 5 4

```

Así, la «nota» que está en lugar 3 es la más pequeña; luego la que está en lugar 1, y así hasta la número 4, que es la mayor.

Esto sirve para ordenar las notas, lógicamente, indexando el vector con estos números de orden:

```
NOTAS[1 2 5 4]
0 3 5 6 8
```

o lo que es lo mismo:

```
NOTAS[NOTAS]
0 3 5 6 8
```

Si queremos colocar NOTAS en orden descendente:

```
NOTAS[NOTAS]
8 6 5 3 0
```

Una airosa aplicación de los operadores ↑ y ↓, que necesitaremos en nuestro último capítulo, es ésta: dada una serie de números:

```
3 8 0 10 9 4 3
```

obtener que puesto ocuparían si estuviesen ordenados, por ejemplo, en forma decreciente; esto es, quién es el primero de la clase; fijémonos, no po-
guertos en orden, sino obtener el orden. Así, la solución sería en este caso:

```
5 3 7 1 2 4 6
```

Puesto que hay 7 elementos, sus números de orden irán del 1 al 7.

¿QUIEN ESTA EL PRIMERO EN LA CLASE?

¿Cómo hacerlo? He aquí el método:

```
S + 3 8 0 10 9 4 3
S[5] + 1 2 3
```

¿Qué hemos hecho? Simplemente, hemos indexado, o sustituido, por la serie de números del 1 al 7 (qs) los elementos de S en el orden que tendrían ordenados, que nos da JS. Querido lector, piénselo un poco. Y quiza se sienta satisfecho conmigo por la elegancia de la solución APL.

DE VIAJE CON LOS HIJOS (PERO A LO POBRE)

Volviendo a nuestras funciones, que no estamos practicando cuanto debieramos, se me ocurre un bonito problema que combina varias cosas aprendidas.

Queremos llevar de viaje a nuestros hijos. Pero como son muchos y costaría mucho dinero llevamos a todos cada vez, establecemos un procedimiento aleatorio para invitar a uno distinto en cada viaje:

```
HIJOS+6 4p'PEPEANA LUISRITAJUANFELI'
HIJOS
```

```
PEPE
ANA
LUIS
RITA
JUAN
FELI
```

Esta variable será utilizada por nuestra función VIAJE:

```
VIAJE
[1] R SORTED AL AZAR SIN REPETICION
[2] AZAR + 6 ? 6
[3] I + 1
[4] N + AZAR[1]
```

```

[5] HIJO ← HIJOS[N];
[6] VIAJE ← (HI), ← HIJO ← HIJO;
[7] I ← I + 1;
[8] ← 49

```

Comentarios:

- línea 1.ª: comentario no ejecutable;
- línea 2.ª: creación de vector de 6 números al azar entre 1 y 6, no repetidos;
- línea 3.ª: creación de «contador» de viajes;
- línea 4.ª: en N ponemos el número del hijo de la serie AZAR, indicado por el «contador» de viajes;
- línea 5.ª: en HIJO ponemos el nombre del hijo a quien le toca este viaje, indexado por N (fila N);
- línea 6.ª: mensaje que dice en qué viaje irá cada hijo;
- línea 7.ª: aumentamos el «contador» para el siguiente viaje;
- línea 8.ª: volvemos a la línea 4.ª, para comenzar de nuevo esas instrucciones.

Al ejecutar, o invocar, VIAJE, veremos qué ocurre:

```

VIAJE
VIAJE: 1 - HIJO: FELI
VIAJE: 2 - HIJO: RITA
VIAJE: 3 - HIJO: ANA
VIAJE: 4 - HIJO: PEPE
VIAJE: 5 - HIJO: JUAN
VIAJE: 6 - HIJO: LUIS
INDEX ERROR
VIAJE[4]
N ← AZAR[1]

```

EL CONTADOR DE VIAJES

¿Qué ha ocurrido? ¡Con lo bien que había salido! Bueno, es lógico. En la línea 7.ª hemos aumentado 1 al «contador» cuando valía 6, y en la línea 4.ª se ha negado a obtener el 7.º elemento de un vector que sólo tiene 6.

¿Qué hacer? Evidentemente, hay que llevar la cuenta y dejar de sumar en cuanto llevemos 6.

Para aprender esto hay que saber otra cosa antes: cómo cambiar el orden de ejecución, y cómo funciona el **operador /** en la función llamada *compresión*.

DE MUDANZA

En primer lugar, sepamos que la flecha

→

indica al intérprete APL que la ejecución del programa, o función, debe continuar en la línea que viene detrás de la flecha. Así:

→ 7

llevará la ejecución a la línea 7.ª, y

→ 0

terminará el programa. Si detrás de la flecha no hay argumento

la ejecución continuará en la línea siguiente. ¿Estamos?

Por otra parte, veamos el **operador compresión, /**. Vimos antes una forma de operar de esta /, cuando a su izquierda había un operador aritmético:

+ / 3 2 - 1

4

VECTORES «LOGICOS»

Esto es, extiende la operación a todo el vector. Pero cuando a la izquierda lo que hay es un vector de 1 y 0 (se les llama vectores lógicos).

```
1 0 1 1 / 2 3 0 B
2 0 B
```

¿Qué ha hecho? Ha seleccionado los elementos del vector si en el argumento izquierdo, en su misma posición, hay 1, y no los toma si hay 0. Es natural que el vector lógico a la izquierda y el argumento de la derecha deberán tener la misma longitud, si no:

```
1 0 1 / 3 4 0 B
LENGTH ERROR
1 0 1/3 4 0 B
A
```

Por cierto, si el argumento izquierdo es 1, se toma todo el argumento derecho:

```
1/2 4 0 B
2 4 0 B
```

y si es 0, el resultado es el vector nulo:

```
0
00/2 4 0 B
```

Se puede aplicar a las matrices, así:

```
1 1 0/2 3 16
1 2
4 5
```

Ha seleccionado las columnas 1 y 2 de la tabla; si queremos seleccionar por filas, escribiremos:

```
1 0/112 3 16
1 2 3
```

como ya sospechaba el lector.

Si el vector izquierdo no es un vector lógico:

```
3 4 4
1 2 0/3 4 0
```

elegirá 1 elemento 3, 2 elementos 4 y 0 elementos 0 (en algunos intérpretes APL daría ERROR).

Ya que estamos tratando los vectores lógicos, vale la pena mencionar a otros 3 operadores más (no asustarse; muy sencillos): los = (igual a), > (mayor que) y < (menor que).

IGUAL, MAYOR, MENOR

Todos ellos comparan argumentos de la misma longitud (lógico) y dan como resultado vectores lógicos: 1, si es verdad; 0 si no lo es:

```
3 4 0 5 = 2 4 0 6
0 1 1 0
4 2 3 > 5 1 6
0 1 0
4 2 3 < 5 1 6
1 0 1
```

¿Ve usted como era sencillo? ¡Pero si no hay nada difícil en APL! Y menos a estas alturas del libro.

Vamos ahora a usar todo lo anterior para resolver por ejemplo, nuestro problema del cambio de ejecución de una línea a otra en forma con-

dicionada, en nuestro caso a que el «contador» I sea mayor de 6. Escríbanos (ya sabemos cómo «abrir» la función):

```

VIAJEC70J
I+I+1
I7J +(7>I+I+1)/4
EBJ [AB]V

```

¿qué quiere decir esto? Veámoslo por partes:

```

7 > I+I+1

```

Cada vez que el ordenador «pasa» por aquí, compara 6 con el valor que forma I después de sumarle 1:

```

7>2 : 1 1 + 1/4 : : + 4
7>3 : 1 1 + 1/4 : : + 4
7>4 : 1 1 + 1/4 : : + 4
7>5 : 1 1 + 1/4 : : + 4
7>6 : 1 1 + 1/4 : : + 4
7>7 : 1 0 + 0/4 : : +

```

al **comprimir** o con 4, el resultado es un vector nulo, como sabemos, así que la ejecución pasa a la línea siguiente; como hemos eliminado la línea 5, la función acaba.

ETIQUETAS

Por cierto, acabamos de ver que la transferencia de ejecución dentro de la función se opera con:

```

+N

```

donde N es la línea donde queremos continuar. Pero en APL es frecuente eliminar o añadir líneas, con lo cual N perdería de inmediato su validez. Para evitarlo se puede poner en su lugar:

```

+LINEA1

```

en que LINEA 1 es una **etiqueta** que se le coloca a la línea de destino, ya no importará que se añadan o quiten líneas. Con esto, nuestra función que dará finalmente así:

```

VIAJECEDJ
I0J VIAJE;AZAR;HIJOS;I;N
I1J R SORTED AL AZAR SIN REPETICION
I2J AZAR←676
I3J I+1
I4J LINEA1←AZARCIJ N←AZARCIJ
I5J HIJO←HIJOSCN;J
I6J *VIAJE: ',*(I)', ' - HIJO: ', HIJO
I7J +(7>I+I+1)/LINEA1
EBJ V

```

VARIABLES PARA ANDAR POR CASA

«¡Ha hecho usted trampa!», dice un lector: «La línea 0 no está como antes, ¿qué ha hecho?».

Pues tiene usted razón, le pido disculpas. Pero me perdonara cuando se lo explique.

En efecto: dentro de la función hemos ido creando variables: AZAR, I, N, HIJO, que luego no se van a necesitar, una vez acabada la ejecución. Esto es, son variables **para uso interno** que lo único que harán será «ensuciar» el WS con cosas inútiles.

Para evitarlo, esas variables internas se escriben (fíjese cómo están escritas, con :J) en la línea 0, y de esta forma se **crean** en el WS mientras la **función se está ejecutando**, pero una vez finalizada desaparecen. ¿Me perdonan?



RESUMEN

En este capítulo hemos aprendido varias cosas:

- cómo sustituir elementos de un vector por otros, mediante la indexación;
- el uso, para ordenar vectores numéricos de los operadores ↑ y ↓ (orden ascendente y descendente);
- cómo usar contadores en una función;
- paso de la ejecución dentro de la función de una a otra línea;
- cómo funciona el operador compresión /, con dos argumentos (el izquierdo, un vector lógico);
- empleo de los operadores =, > y <;
- etiquetas identificadoras de línea;
- designación de variables internas a la función como variables locales.

Dialoguemos con el ordenador por la ventana.—Un zahori algo tramposo.—El psiquiatra en el ordenador.—Siempre con el ejemplo.

N

O sé si el lector habrá pensado que, hasta ahora, todos los datos que el ordenador ha usado en nuestros programas o funciones le venían «impuestos», ya estaban en el WS antes de empezar a ejecutarse.

Y, sin embargo, estamos viendo todos los días en la vida real cómo los usuarios se pasan el tiempo «escribiendo» datos en teclados para introducirlos en sus terminales o en sus ordenadores. Como la azafata en el aeropuerto reclea el nombre del viajero y el ordenador responde si tiene billete conforme, a qué hora sale su vuelo, etc.

Efectivamente: tenemos que encontrar la forma de «dialogar» con el ordenador. Hacerle preguntas, hacer que nos pida datos y nos dé respuestas. Ello se consigue en APL (entre otros) con la llamada «ventana», o «quad».

LA VENTANA INDISCRETA

Así, si en una instrucción escribimos:

D

el ordenador queda esperando a que escribamos **nosotros**, con el teclado, algo. Y ese algo lo utilizará el ordenador, por ejemplo, para colocarlo en una variable.

D:

RESPUESTA ← D

```
'COSAS'  
RESPUESTA  
COSAS
```

Al «abrirse» la «ventana» el ordenador lo indicó así:

```
0:
```

y se quedó esperando; hemos escrito 'COSAS' y este vector de caracteres se ha introducido en la variable RESPUESTA. ¿De acuerdo?
Normalmente, en realidad, se emplea para solicitar información numérica, no alfanumérica:

```
N+0  
01  
3 2 5  
N  
3 2 5
```

y el (ventana-comilla) para pedir información literal:

```
NOMBRE + 0  
LUIS  
NOMBRE  
LUIS
```

Hay aquí, como vemos, algunas diferencias:

— En primer lugar, **no se muestra el** , el ordenador muestra una línea vacía en la cual tenemos que escribir nuestra respuesta literal **sin comillas**; ella se asignará a la variable NOMBRE en ese momento. Así, mientras antes aparecer:

```
0:
```

y el ordenador esperaba respuesta en la siguiente línea, aquí simplemente aparece una línea vacía para la respuesta.
Por otra parte, si escribimos:

```
LUIS  
0 + NOMBRE
```

esta es la forma (una forma) que tiene de comunicarnos información el ordenador. Claro que, más sencillo, habría sido:

```
LUIS  
NOMBRE
```

Pero eso no es posible cuando para abreviar espacio, por ejemplo, tenemos varias instrucciones en una línea:

```
I + 3, 0 p 0 + NOMBRE
```

HAY QUE AHORRAR

¿Qué hacemos aquí? En una línea:

— hemos mostrado en la pantalla el contenido de la variable NOMBRE;

— se ha creado un vector nulo Op NOMBRE;
— se ha «catenado» 3 y un vector nulo; el resultado seguirá siendo 3;
— se ha asignado ese 3 a la variable I.
¿De acuerdo?

Cuando usamos para comunicación se salta una línea: si usamos

```
0:  
LUIS  
0 + NOMBRE
```

y el ordenador queda esperando la siguiente instrucción en la misma línea en que está LUIS.

Y eso ¿para qué sirve? Bueno: sólo para conseguir determinados efectos más o menos estéticos en el «diálogo» con el ordenador. En todo caso, no sirve para que el ordenador «comunique» valores numéricos; sólo para literales. En cambio si los acepta ambos.

EL ZAHORI TRAMPOSO

Ya es hora de que pongamos algunos ejemplos. Vamos a «inventar» una función algo tonta, pero que aclara el uso de y .

```

VADIVIND2.11
E01 A DIVIND;N;P
E11 A USD DEL D Y DEL B
E21 L: SU NOMBRE, POR FAVOR
E31 *NACIMIENTO; ,Q;N+0
E41 P+1986-4#
E51 *LAS ESTRELLAS DICEN, ' ;N
E61 *QUE VD. TIENE DE EDAD ; ,*P
E71 *L: OPODE EL SIGUIENTE
  
```

Comentarios:

- línea 2.ª: etiqueta L; mensaje;
 - línea 3.ª: pide datos alfanuméricos; los pone en N; crea un vector nulo; lo catena al mensaje;
 - línea 4.ª: pide dato alfanumérico; lo «ejecuta» y transforma en número; lo resta de 1986 y lo asigna a P;
 - línea 5.ª: mensaje catenado al nombre;
 - línea 6.ª: mensaje catenado con P (formateado);
 - línea 7.ª: mensaje del ordenador, convertido en vector nulo, que se catena a la etiqueta L; el ordenador ejecuta la línea L a continuación.
- Si ejecutamos la función:

```

ADIVIND
SU NOMBRE, POR FAVOR
PEPE
NACIMIENTO
1950
LAS ESTRELLAS DICEN, PEPE
  
```

```

QUE VD. TIENE DE EDAD 36
EL SIGUIENTE
SU NOMBRE, POR FAVOR
  
```

La función continuará indefinidamente, pues no se ha previsto cómo interrumpirla. Para ello:

```

VADIVIND2.11
E2.11 + ('0' = 1 + N + B) / 0
E2.21L3J *NACIMIENTO; ' V
  
```

Con esta modificación en la nueva línea 2.1 (que al cerrar la función quedará como línea 3) se pide una entrada alfanumérica, el nombre y se asigna a N; se toma el primer carácter y se compara con '0'; si es igual resultará 1 y $\rightarrow 1/0$ quedará $\rightarrow 0$, o sea, que se acabará la ejecución de la función. Así, cuando respondamos a la pregunta del nombre con 0, haremos hecho terminar la ejecución.

EN EL MANICOMIO

Otro ejemplo de «aplicación» de la capacidad de diálogo del ordenador a través de la «ventana» lo constituye el siguiente en que se simula una conversación entre «paciente» y «psiquiatra»:

```

OMINIELISAC01V
E01 MINIELISA;N;W
E11 *SOY LA DOCTORA ELISA, COMO SE LLAMA VD.?
E21 L0: + ('0' = 1 + N + B) / L1
E31 *BUENDOS DIAS, ' ;N, ' . COMO SE ENCUENTRA HOY?
E41 W+B
E51 L3: *YA, ALGUN COMENTARIO ADICIONAL?
E61 + ('0' = 1 + W + B) / L2
E71 + ('N' = 1 + W) / L2
E81 +L3
E91 L1: +L0, 0, 0 + ' DEBE ESCRIBIR SU NOMBRE, POR FAVOR'
E10J L2: *VED QUE NO DESEA HABLAR. ADIDS, ' ;N
  
```

Comentarios:

- línea 1.^a: mensaje;
- línea 2.^a: si el primer carácter de la entrada literal es un blanco la ejecución se transfiere a la línea L1;
- línea 3.^a: mensaje con el nombre del «paciente»;
- línea 4.^a: la entrada literal se «almacena» en la variable W, que no se usa para nada (quizá, como ocurre en la realidad, con perdón de los psiquiatras);
- línea 5.^a: mensaje;
- línea 6.^a: si el «paciente» contesta en blanco, la función sigue en L2;
- línea 7.^a: si contesta «no» sigue en L2;
- línea 8.^a: en otro caso, la función vuelve a L3;
- línea 9.^a: L1: pide que se escriba el nombre y vuelve a L0;
- línea 10.^a: L2: mensaje y cierre de la función.

Si ejecutamos:

```
MINIELISA
SOY LA DOCTORA ELISA. COMO SE LLAMA VD.?
DEBE ESCRIBIR SU NOMBRE, POR FAVOR
PEPE
BUENOS DIAS, PEPE. COMO SE ENCUENTRA HOY?
MAS O MENOS
YA. ALGUN COMENTARIO ADICIONAL?
SI, QUE ESTOY HARTO
YA. ALGUN COMENTARIO ADICIONAL?
YO QUE SE
YA. ALGUN COMENTARIO ADICIONAL?
NO, NINGUNO
VED QUE NO DESEA HABLAR. ADIOS, PEPE
```

Es simpático ¿no? Pues hay un programa real, llamado precisamente ELISA, más complejo y elaborado por supuesto, que en esencia hace lo mismo: simula una conversación con frases «universales» de aliento del doctor para que el paciente siga su monólogo hasta que se desahogue.

RESUMEN

En este capítulo hemos desarrollado problemas prácticos de la escritura de funciones, como:

- mensajes del operador;
- mensajes del ordenador;
- modos de escribir instrucciones en una sola línea;
- simulación de diálogos.

CAPITULO 1

Sigamos con las cosas lógicas: operaciones booleanas (que meten miedo).—Los soldados que pasan debajo de la vigila.—Más difícil todavía: los soldados de permiso (los cinco primeros y los cinco últimos).—Por el techo y por el suelo.

U/ANDO, en anterior capítulo, hemos comparado por =, > y < dos argumentos, vimos que el resultado es un vector de 0 y 1, un vector lógico:

```
4 5 0 6 > 3 1 2 0
1 1 0 1
```

Estos vectores lógicos se emplean en APL en varias aplicaciones; por ejemplo, para seleccionar elementos de un conjunto. Queremos saber qué elementos de un vector son iguales a un cierto valor:

```
V + 5 4 6 5 3 6
+ M + V = 5
1 0 0 1 0 0
```

Con M podemos seleccionar los elementos del vector V:

```
..... M/V
5 5 .....
.....
2 ..... +/M
```

dice que el número de elementos iguales es 2.

BUUU...LI

Entre vectores lógicos se aplican las llamadas **operaciones booleanas**, en las que los vectores se comparan elemento a elemento por \wedge (y) o por \vee (o). Así, si dos elementos son 1, la respuesta es 1 por \wedge .

$$\begin{array}{l} \text{a)} \quad 1 \wedge 1 \\ \quad \quad 1 \\ \text{b)} \quad 0 \wedge 1 \\ \quad \quad 0 \end{array}$$

Esto es, en el caso a), la condición 1 se cumple en ambos; en el caso b), así no cumplirse en 1 ya no se cumple **en los dos**.

En cambio por el operador \vee (o) queremos que la condición positiva, la condición 1, se cumpla al menos en uno de los dos elementos:

$$\begin{array}{l} 1 \vee 0 \\ 1 \\ 1 \vee 1 \\ 1 \\ 0 \vee 0 \\ 0 \end{array}$$

¿O? ¿Y?

En el último ejemplo, al no cumplirse en **ninguno** de los dos elementos, el resultado de la comparación o es 0. Más completo:

$$\begin{array}{l} 1 \quad 0 \quad 1 \quad 1 \quad \wedge \quad 0 \quad 0 \quad 1 \quad 1 \\ 0 \quad 0 \quad 1 \quad 1 \\ 1 \quad 0 \quad 1 \quad 1 \quad \vee \quad 0 \quad 0 \quad 1 \quad 1 \\ 1 \quad 0 \quad 1 \quad 1 \end{array}$$

¿Para qué sirve? Tengamos una «respuesta» a una pregunta, cual es el cuadrado de 4?

RESP = 16

Al usar la ventana alfanumérica la respuesta será 16 (dos caracteres). Hay que ver si no se ha equivocado el examinando y comparar con la respuesta correcta:

$$\begin{array}{l} 16 \\ 1 \quad 1 \end{array} \quad \text{'16'} = \text{RESP}$$

¿Cómo sabemos que la respuesta es 11?

$$\begin{array}{l} 1 \\ \wedge / 1 \quad 1 \end{array}$$

(extendemos la operación \wedge a todo el vector; $1 \wedge 1$)

Si el resultado es 1, la respuesta era correcta; si hubiere sido, por ejemplo, 12:

$$\begin{array}{l} 16 \\ 1 \quad 0 \\ \wedge / 1 \quad 0 \\ 0 \end{array} \quad \text{'16'} = \text{'12'}$$

EL INQUISIDOR DE RESPUESTAS

Escribiremos, pues:

$$\wedge (\wedge / \text{'16'} = \text{RESP} \wedge 0) / \text{SEGUIR}$$

Si la respuesta es correcta, el parentesis valdrá 1, y por comprension la ejecucion se trasladará a SEGUIR:

SEGUIR

En caso contrario irá a la siguiente instruccion donde habrá un mensaje de que está mal, etc.

Para preguntas literales, se hará lo mismo. ¿Cuál es la capital de Francia?

R + (A/7PARIS' ≠ R + E)/ERROR

En este caso hemos comparado por ≠ (no igual); si el resultado del parentesis es 1, esto es «PARIS» no es igual a la respuesta, iremos a una línea donde se tratan las respuestas incorrectas. ¿De acuerdo?

¿CUANTOS APROBARON?

Ya estamos terminando la parte expositiva de este libro; apenas un par de operadores más y estaremos listos para los capítulos finales, donde haremos un par de programas más complejos. Pero antes vamos a ver un bonito ejemplo de aplicación de algunas cosas anteriores.

En una clase, los alumnos tienen un examen:

NOTAS + 5 4 3 2 6 10 0 2

() para hacerlo de verdad complejo, creamos las notas en forma aleatoria:

NOTAS + -1 + 2 50 2 11

(Las notas deben ir del 0 al 10 en esta clase de 50 alumnos; al tener un vector de 50 números 11, los aleatorios irán del 1 al 11; al sumar -1 las notas «inventadas» irán del 0 al 10. ¿Está ahora claro?).

¿Cuántos aprobaron? (5 es aprobado, por ejemplo). Nada más fácil:

28

+ /4 < NOTAS

Es más fácil que contarlos ¿no?

LOS CABOS GASTADORES

Otro ejemplo. En la Caja de Reclutas se quiere mandar a un regimiento especial a todos los soldados cuya altura pase de 1.75 m. Los soldados están numerados del 1 al 100. Queremos saber los números de los soldados a quienes hay que enviar.

ALTURA + 150 + ? 100 2 50

(imaginamos, para «inventar» el vector de alturas, que la menor posible es 150 cm. y la máxima 200 cm.).

LIMITE + 175

PASAN + LIMITE < ALTURA

Pero PASAN es un vector lógico (1 y 0), 1 para quienes tienen su altura por encima del límite y 0 en caso contrario. ¿Cómo «sacar» los números de los soldados con 1? Muy fácil:

52

+ /PASAN

PASAN/ALTURA
2 3 5 10 11 12 13 14 16 19 23 27 29 30 31 37 40 43 50 54 55 56 58 59 61 63 64
65 66 68 69 71 72 73 74 75 76 78 79 82 83 85 86 87 88 89 90 92 93 94 98
99

o si no sabemos que son 100,

PASAN/VAALTURA

¿Qué tal?

Vamos para terminar, a volver a un pobre operador a quien tenemos completamente abandonado, aunque lo hemos mostrado fugazmente e incluso utilizando sin haberle presentado en regla el operador ↑ (tomar), y su homólogo el operador ↓ (quitar).

Tomar, ↑, como ya vimos por encima, selecciona del argumento derecho tantos elementos como indique el argumento izquierdo:

```
4 ↑ 2 3 0 -1 2 4 6
2 3 0 -1
```

¿Y si el argumento izquierdo es negativo?

```
-2 ↑ 2 3 0 -1 2 4 6
4 6
```

Evidente: «tomar» 2, pero del final. Si el argumento izquierdo es 0:

```
0 ↑ 2 3
```

Eso es: un vector nulo. Queda claro: ↑ y ↓ siempre producen en su aplicación vectores.

DE QUITA Y PON

Veamos el operador ↓ (quitar):

```
4 ↓ 2 3 0 -1 2 4 6
2 4 6
```

Como ↑ se «deja» abandonados a los cuatro primeros, sólo «sobreviven» los tres últimos:

```
-3 ↓ 2 3
```

(Vector nulo).

Una particularidad:

```
5 ↑ 3 2
3 2 0 0 0
```

Esto es, el número de elementos **tomados** se completa con ceros:

```
4 ↑ 'PEPEANITA'
PEPE 'BT'ANITA'
ANITA
```

No se ve quizá, pero hay un desplazamiento a la derecha de ANITA en 3 blancos, los que se han añadido para completar los 8 pedidos por el argumento -8.

En matrices hay que agudizar la atención:

```
1 2
6 7
TABLA + 4 Sp,20
2 2 ↑ TABLA
```

Ha tomado dos filas y dos columnas:

1	2	3	4	5
6	7	8	9	10
11	12	12	14	15
16	17	18	19	20

Si ponemos:

11 12 2 2 1 TABLA
15 17

Con el operador ↓ (quitar):

2 2 1 TABLA
13 14 15
18 19 20

(imaginemos que hemos «borrado» las dos primeras filas y las dos primeras columnas: ¿qué quedaría?).

También:

3 0 1 TABLA

(hemos tomado tres filas pero ninguna columna):

3 0 1 TABLA
1 2 3 4 5

Querido lector: hasta aquí la presentación de esta «introducción» al lenguaje APL. Quedan, como luego comentaremos, algunos, quizá muchos, aspectos por tratar, pero no esenciales para iniciar una relación afectiva personal, con esta maravillosa y potente herramienta de programación. En los dos próximos capítulos haremos unos programas de aplicación que servirán como resumen y broche final al libro.

RESUMEN

El capítulo 11 del libro ha terminado de exponer algunos operadores útiles:

- se ha insistido en los vectores lógicos y sus aplicaciones añadiendo los «booleanos»: A y V;
- se ha examinado, mediante esos vectores lógicos, el problema de «verificar» la exactitud de las respuestas dadas al ordenador;
- por último, se han visto los operadores ↑ (tomar) y ↓ (quitar), tanto con vectores como con matrices.

CAPITULO 12

Programa para el cálculo de los valores dietéticos de una receta culinaria.

U

NA receta es, simplemente, la cuantificación de los elementos que entran en la preparación de un plato.

Así, habrá que prever una tabla en que tengamos los ingredientes de varios tipos y sus valores dietéticos; tabla que luego será «consultada» para usar esos valores en el cálculo alimenticio de la receta.

La tabla, que llamaremos INGR, deberá tener estos elementos (entendemos que en una tabla más completa debería haber más elementos):

- 1 Número del ingrediente.
- 2 Calorías por gramo.
- 3/5 Vitaminas A, B, C, D, etc., por gramo.
- 6/7 Minerales Fe, Ca, por gramo.
- 8/10 Minerales, grasas, hidratos de carbono, por gramo.

INGR ← 0 10p0

INGR será la matriz de valores numéricos, creada con 0 filas (para añadirlo) y 10 columnas:

NOMINGR ← 0 15p1

NOMINGR será la matriz de nombres de los ingredientes, para los cuales dejamos 15 columnas.

Veamos la función que añade ingredientes:

```

VINGREDIENTEEDIV
EQ3  INGREDIENTE;N;C1;C2;C3;C4;C5;C6;C7;C8;C9
C13  L1+(C10=1)+N+D,0pD+VINGREDIENTE (0=FIN):?)/0
C23  NOMINGR+NOMINGR,(1115)+N
C33  C1+D,0pD+CALORIAS/GR:
C43  C2+D,0pD+VITAMINA A/GR:

```

```

E5J  C3+D,0pE+VITAMINA B/GR:
E6J  C4+D,0pD+VITAMINA C/GR:
E7J  C5+D,0pD+MILIGR FE/GR:
E8J  C6+D,0pE+MILIGR CA/GR:
E9J  C7+D,0pE+MILIGR PRDTEINAS/GR:
E10J C8+D,0pE+MILIGR GRASAS/GR:
E11J C9+D,0pD+MILIGR HIDRATOS DE CARBONO/GR:
E12J N+1↑/INGR
E13J +L,0pINGR+INGR, E11 (N+1), C1, C2, C3, C4, C5, C6, C7, C8, C9

```

Comentarios:

— línea 1.ª, una línea compleja, que da un mensaje con la opción 0 para terminar; en la misma línea se da el mensaje por una ventana que salta línea, y se coloca en N el nombre del ingrediente, o se termina si se introduce 0);

— línea 2.ª, se toman 15 caracteres del nombre N, y se catenan a la matriz NOMINGR;

— líneas 3.ª a 11.ª, piden los valores numéricos de los diversos elementos y los colocan en las variables C1 a C9;

— línea 12.ª, interesante; N (que ya podemos emplear para otro uso), número de líneas de la matriz INGR como estaba antes de añadir los datos actuales; se obtiene como primer elemento (1↑) del p de INGR, como es lógico;

— línea 13.ª, catenamos a la matriz el vector de todos los valores aliménticios, encabezado por el número del ingrediente (el anterior más 1); se transfiere la ejecución a la etiqueta L de la línea 1, para que vuelva.

Al ejecutar la función:

```

INGREDIENTE
INGREDIENTE (O=FIN):
HIGADO CERDO
CALORIAS/GR:
D:
1.6
VITAMINA A/GR:
B:
23
VITAMINA B/GR:
D:
78
VITAMINA C/GR:
D:
4

```

```

MILIGR FE/GR:
D:
12
MILIGR CA/GR:
B:
23
MILIGR PRDTEINAS/GR:
D:
560
MILIGR GRASAS/GR:
D:
250
MILIGR HIDRATOS DE CARBONO/GR:
D:
39
INGREDIENTE (O=FIN):
0

```

```

RECETA
RACIONES EN LA RECETA:
D:
4
NUM DEL INGREDIENTE (O=FIN):
D:
1
CANTIDAD DEL INGREDIENTE, EN GR:
D:
200
NUM DEL INGREDIENTE (O=FIN):
D:
2
CANTIDAD DEL INGREDIENTE, EN GR:
D:
40
NUM DEL INGREDIENTE (O=FIN):
D:
4
CANTIDAD DEL INGREDIENTE, EN GR:
D:
500
NUM DEL INGREDIENTE (O=FIN):
D:
0

```

Después de introducir este ingrediente, cerramos la función; supongamos ya un poco más «llenas» ambas matrices, con varios ingredientes (son valores ficticios; en libros de dietética se encontrarán los reales):

NOMINGR
PATATA
ACEITE
CARNE VACA

7	2*INGR	12.00	34.00	15.00	12.00	25.00	12.00	14.00
1.00	1.25							
780.00								
2.00	2.12	.00	.00	.00	.00	.00	.00	790.00
.00								
3.00	1.50	12.00	45.00	8.00	5.00	4.00	250.00	120.00
180.00								

Veamos ahora la preparación de las recetas:

```

*RECETACD]
[01] RECETA:N;C;R
[11] D+9*0
[21] LO:R+D,Q*0+*RACIONES EN LA RECETA:
[31] L1:(O=N+D,Q*0+*NUM DEL INGREDIENTE (O=FIN):*)/O
[41] C+D,Q*0+*CANTIDAD DEL INGREDIENTE, EN GR:
[51] O+O+Cx1+, INGRFN;J+R
[61] +L1
  
```

Comentarios:

- línea 0: no hemos puesto como variable local a O, porque necesitamos que su valor se quede en la WS, como veremos;
- línea 1: creamos el vector O de nueve 0, donde iremos acumulando las calorías, vitaminas, etcétera, de los diversos ingredientes;
- línea 2: colocamos en R el número de raciones que se preparan con la receta;
- línea 3: en N colocamos los números de los distintos ingredientes utilizados (debemos tener la lista NOMINGR delante de nosotros);
- línea 5: acumulamos en O el vector Cx INGR [N:] + R que da para el ingrediente N y la cantidad C (dividiendo por el número de raciones) los valores alimenticios que aporta a la receta;
- línea 6: volvemos a L1 (línea 3) hasta que no haya más ingredientes.

En Q tenemos los valores alimenticios que aporta por persona la receta; para hacerlo más visible podemos añadir como última línea de RECETA una función auxiliar, MOSTRAR:

```

*MOSTRARCD]
[01] MOSTRAR
[11] *VALORES DIETETICOS CALCULADOS, POR RACION:
[21] *CALORIAS:
[31] *VITAMINA A:
[41] *VITAMINA B:
[51] *VITAMINA C:
[61] *HIERRO:
[71] *CALCIO:
[81] *PROTEINAS:
[91] *GRASAS:
[10] *HIDRATOS DE CARBONO:
  
```

Evidentemente, si se incluye:

```

MOSTRAR
VALORES DIETETICOS CALCULADOS, POR RACION:
CALORIAS: 282.7
VITAMINA A: 3475
VITAMINA B: 11450
VITAMINA C: 1250
HIERRO: 2100
CALCIO: 4025
PROTEINAS: 70600
GRASAS: 39850
HIDRATOS DE CARBONO: 43875
  
```

dentro de RECETA, no hace falta tener Q como variable global en el WS; se deberá entonces asignarla como variable local, añadiéndola a la línea 0:

```

*RECETACD]
[01] RECETA:N;C;R;Q
[11] Q+5*0
[21] LO:R+D,Q*0+*RACIONES EN LA RECETA:
[31] L1:(O=N+D,Q*0+*NUM DEL INGREDIENTE (O=FIN):*)/L2
[41] C+D,Q*0+*CANTIDAD DEL INGREDIENTE, EN GR:
[51] O+O+Cx1+, INGRFN;J+R
[61] +L1
[71] L2:MOSTRAR
  
```

Programa para crear el fichero de calificaciones en un

Este programa se podría hacer más completo y complejo, con un «archivo» de recetas, para confeccionar «menús» completos, y optimizarlos por combinaciones varias entre primeros y segundos platos, bebida y postre, ajustándolos a las necesidades humanas (incluso de niños separadamente de adultos, por ejemplo)... Pero eso también es, como antes dijimos, fijar el rizo, y no es más APL, sino más trabajo.

E

N el programa (conjunto de funciones y variables) que cierra este libro de introducción al lenguaje APL, presentaremos:

- la forma moderna de ejecutar aplicaciones, por medio de «menús»;
- algunos nuevos métodos de desviar la ejecución de la función de una a otra línea;
- usos varios del importante operador \uparrow (ejecutar).

El programa que llamaremos FICHERO, contiene variables y funciones. Veámoslas:

```

VARIABLES
AGREGAR  CALCULO  CALIFICAR  CAMBIAR  ELIMINAR  FICHERO  LISTAR
VARS
CLASE MENU ME1 ME2 NOTAS
    
```

Entre las variables, merece la pena fijarse en ME1 y ME2:

```

ME1
NO EXISTE ESE ALUMNO. DE NUEVO, POR FAVOR
ME2
ELIJA UN NUMERO DE LOS MOSTRADOS
    
```

Son, como se entiende fácilmente, mensajes. ¿Por qué ponerlos en variables y no dentro de las funciones? Por varias razones:

- si se usan en varias funciones, evitamos escribirlos una y otra vez;
- dan un aspecto más compacto a las funciones;
- permiten modificación directa, sin tener que abrir las funciones;
- en fin, para traducir a otras lenguas, es mucho más simple.

Vale la pena hacer referencia al Apéndice C, en que damos algunos consejos sobre la forma de programar APL; entre ellos, puede decirse que las **variables globales** deberían nombrarse con tres letras/números (ocupan mucha menos memoria) y las variables internas a las funciones o **variables locales**, con una letra.

Otra variable en FICHERO es:

```

MENU
AGREGAR      : 1
CAMBIAR      : 2
CALIFICAR    : 3
ELIMINAR     : 4
LISTAR       : 5
FIN          : 0
  
```

Es una matriz en la cual se muestra las diversas opciones del programa y la opción FIN, usualmente numerada como 0 y al final de la tabla; fijémosnos que los : están agrupados en la columna 13 y no sólo por razones estéticas; lo veremos luego.

En las variables **CLASE** y **NOTAS** se irán colocando los alumnos y sus calificaciones con las funciones que luego veremos (se crean inicialmente 0 24 y 0 6, la primera alfanumérica y la otra numérica pues tendrá notas, promedios y rangos).

Veamos las funciones del programa; la principal, que invoca a todas las demás es:

```

VFICHERO[D1]V
FICHERO;N
R PREPARA EL FICHERO DE CALIFICACIONES DE UNA CLASE
L0; ' ', OpD+0 P C I O N E S ', OpD+ ' '
E31 MENU
E41 *ELIJA POR SU NUMERO UNA DE LAS OPCIONES: ', OpD+ ' '
E51 *(+/1 2 3 4 5 0=N+0)/L1
E61 +L0, OpD+ME2
  
```

```

E71 L1:+(0=N)/L2
E81 *L2, MENU;J
E91 +L0
E101 L2: NO OLVIDE HACER )SAVE SI HUBO CAMBIOS
  
```

Comentarios:

- línea 1.^a: resumen de su objetivo;
- línea 2.^a: etiqueta L0; salta línea; mensaje; salta línea (no olvidemos que se ejecuta de derecha a izquierda);
- línea 3.^a: muestra la tabla MENU;
- línea 4.^a: salta línea; mensaje;
- línea 5.^a: pide la variable numérica N y la compara por = a las respuestas validas al MENU (0, 1, 2, 3, 4, 5); si la respuesta es uno de esos números, el vector lógico que resulta tendrá un 1, y al sumarlo (0 1 0 0 0 0, por ejemplo) quedará seleccionada la línea L1, que es la 7.^a donde continúa la función; en caso contrario sigue la ejecución en la
- línea 6.^a: muestra el mensaje ME2, salta a la línea L0 de nuevo;
- línea 7.^a: si la opción elegida fue 0 salta a la línea L2;
- línea 8.^a: importante línea que muestra cómo usar el **operador ejecutar**. Selecciona la línea del menú N (opción elegida); la convierte en vector con la, (coma); toma los 12 primeros caracteres (para eliminar los : y el número de opción); ahora tenemos un vector con el nombre de lo que queremos hacer. Pero como en el programa **hay una función con ese nombre** (ver la lista), si **ejecutamos** esa palabra se convertirá en objeto APL, esto es, se ejecutará la función que hace lo que dice la palabra;
- línea 9.^a: una vez ejecutada la opción viene a esta línea 9 al orden de ejecución y se transfiere a la línea L0, que vuelve a mostrar el MENU, etc.
- línea 10.^a: etiqueta L2, muestra un mensaje muy importante, para que no se olvide al usuario de guardar en el disco/diskette el WS, si hubo cambios (nombres y notas cambiados o añadidos, etc.).

Vamos a ver a continuación las distintas funciones que hacen los trabajos que ofrece MENU:

```

VAGREGAR[D1]V
FICHERO;N
R AGREGA ALUMNOS A CLASE
L0:+(?0'=1+N+0, OpD+ ' NUEVO ALUMNO (0=F IN) ' )/0
E31 CLASE+CLASE, L1124+N
E41 NOTAS+NOTAS, L115+0
E51 +L0
  
```


Comentarios:

- línea 2.^a: pide el nombre de un nuevo alumno, salta línea y abre el teclado para que el nombre se coloque en N; como hemos indicado que **para terminar se teclée 0**, compara el **primer carácter** de N con el literal «0» (no el número 0, cuidado); si son iguales, resultará 1, y se transferirá la ejecución a 0, o sea, termina la función.
- línea 3.^a: como la tabla CLASE de nombres de los alumnos tiene 24 columnas, se catenan 24 caracteres de N a CLASE por las flas;
- línea 4.^a: habrá también que ampliar la tabla NOTAS, colocando 5 ceros (ya se pondrán las notas, etc., mas tarde: no es que hayamos decidido suspender «a priori» al pobre alumno añadido);
- línea 5.^a: volvemos a L0 para seguir añadiendo alumnos.

Otra función, opción del MENU número 2:

```
VCAMBIARL010
[0] CAMBIAR:L:N
[1] A CAMBIA LINEAS EN CLASE
[2] L+1/PCLASE
[3] CLASE,' ',*(L,1)P\L
[4] LO:+(O=N+0,OPD+'NUMERO DE ALUMNO A CAMBIAR (0=FIN)')
) /O
[5] *(L<N) /? +LO,OPD+ME1'
[6] CLASEIN;1
[7] CLASEIN;1+241B
[8] +LO
```

Comentarios:

- línea 2.^a: se almacena en L el número de líneas de la tabla CLASE;
- línea 3.^a: se muestra la tabla CLASE, catenada con otra tabla, una columna de números para identificar a cada alumno; fíjese el lector que esa tabla de L columnas ha de **formatearse** para convertir los números en letras; catenamos también un blanco ' ' para que haya espacio entre nombres y números de orden;
- línea 4.^a: mensaje semejante a la línea 2 de AGREGAR, N lo pedimos numérico;
- línea 5.^a: nueva forma de transferencia **con mensaje incluido**; comparamos N (número de alumno a cambiar) con L (número de alumnos en la tabla); si este es **menor** del número pedido, es un error del usuario; la respuesta a la comparación será 1, y por comprensión resulta el vector de tras de /; es un vector alfanumérico letras, en suma. Por eso, lo **ejecutamos** y se convierte en objeto APL, que: a) muestra un mensaje; b) transferirá la ejecución a la línea L0 de nuevo;

- línea 6.^a: muestra al alumno seleccionado para cambio en la tabla CLASE;

— línea 7.^a: pide por ventana el cambio en el nombre, toma 24 caracteres y lo coloca en la tabla CLASE en la posición que habíamos seleccionado; atención: aunque no se quieran hacer cambios, hay que escribir el nombre de nuevo porque en caso contrario se colocarían blancos en la tabla (hay formas de programar para evitarlo, pero por ahora no se mostrarán).

Opción número 3:

```
VCALIFICARL010
[0] CALIFICAR:N;A;L
[1] A AGREGA/CAMBIA NOTAS, CALCULA MEDIAS Y ORDEN
[2] L+1/PCLASE
[3] CLASE,' ',*(L,1)P\L
[4] LO:+(O=N+0,OPD+'NUM ALUMNO A CALIFICAR (0=FIN)')P\L3
[5] *(L<N) /? +LO,OPD+ME1'
[6] CLASEIN;1,NOTASIN;1
[7] L1:+(S=PA+2510,OPD+'NUEVAS NOTAS (3)
[8] +L1,OPD+'3 NOTAS, POR FAVOR'
[9] L2:+LO,CANOTASIN;1+A,0 0
[10] L3:CALCULO
```

Comentarios:

- líneas 2.^a a 5.^a: como en CAMBIAR;
- línea 6.^a: muestra el nombre del alumno y sus calificaciones actuales;
- línea 7.^a: mensaje con blancos para que las notas que se coloquen en A caigan debajo de las actuales; fijémonos que **no salta** línea, por lo cual hay que quitar de la entrada literal 25 caracteres, los del mensaje; si **no el mensaje entraría en A con las notas**; en A, al ejecutar el mensaje, entran números; se compara su número (pA) con 3, que son las notas pedidas; si es correcto se va a la línea L2; si no, a la
- línea 8.^a: mensaje de error, y vuelta a la línea L1;
- línea 9.^a: como en NOTAS tenemos 5 números, no 3 como hemos puesto en A, se catena 0 0 a A antes de colocarlo en NOTAS, en el número de orden del alumno; se vuelve a L0;
- línea 10.^a: una vez introducidas las tres notas de todos los alumnos, o incluso aunque sólo haya habido en algunas de aquellas, se ejecuta una función que calcula las notas medias y el rango en la clase de cada alumno.


```

VCALCULO:CLD3V
E01 CALCULO: MEDIAS
E11 A CALCULA NOTA MEDIA Y RANGO EN NOTAS
E21 GPP+3
E31 MEDIAS+NOTAS/4+0.1*TO.05+10*(1-3)*+/NOTAS/4.31
E41 MEDIAS+MEDIAS1+1.4MEDIAS
E51 NOTAS/5+MEDIAS

```

Comentarios:

- línea 2.ª: reducidas las cifras significativas mostradas a 3;
- línea 3.ª: colocamos en MEDIAS la 4.ª columna de NOTAS; en esta ponemos el promedio de las 3 calificaciones, 3 primeras columnas de NOTAS, redondeado (se multiplica por 10, se suman 5 centésimas, se haya el techo y se divide de nuevo por 10);
- línea 4.ª: se obtiene el orden (ver capítulo 9), o rango;
- línea 5.ª: se coloca el rango en la columna 5 de NOTAS.

Una importante función cuando se manejan ficheros es la que da la posibilidad de:

```

VEELIMINARL3V
E01 ELIMINAR LINEA
E11 A ELIMINA LINEAS EN CLASE Y EN NOTAS
E21 L+1+CLASE
E31 CLASE+' '+*(L-1)PVL
E41 LOG+LOG+D,OPD+*NUMERO DE ALUMNO A ELIMINAR (D=FIN)*/0
E51 # (LEN)/7*ALO,OPD+MEI
E61 +(*S/11B,OPD+TECLEE "SI" SI QUIERE ELIMINAR A ...CLASE
E71 ENJ)/LO
E81 LIA+PVL
E91 +LO,OPCLASE+A/11CLASE),OPNOTAS+A/11NOTAS

```

Comentarios:

- líneas 2.ª a 5.ª: como en CAMBIAR y CALIFICAR;
- línea 6.ª: mostramos el nombre del alumno que se quiere eliminar para solicitar **confirmación** de eliminación por si es un error o nos arrepentimos; por SI (mejor dicho, por S, primera letra de la contestación, se quitamos en LO), si no, pasamos a la
- línea 7.ª: LI: creamos un vector lógico A con unos y un 0 en el orden de la línea que se quiere eliminar (por cierto, empleamos un operador no mostrado) hasta ahora, # (no igual que funciona como el = dando

un vector lógico de 1 si se cumple la condición de desigualdad y de 0 de lo contrario (de no se cumple o sea, donde son iguales);

— línea 8.ª: con el vector A «comprimimos» las tablas CLASE y NOTAS, para eliminar la línea que no queremos. Observe el lector que para que estén en una línea, las convertimos en **vector nulo**, como antes hicimos. Pero si no ponemos CLASE y cuanto la acompañamos entre paréntesis no funcionará ya que estamos concatenando CLASE (matriz) con un vector (nulo), lo cual no es posible. Una vez eliminada la línea y asignadas las tablas a su antiguo nombre, ahora con una línea menos, se vuelve a LO, para recomenzar la función.

Nos queda una sola función por examinar:

```

VELISTARL3V
E01 LISTAR:W
E11 A LISTA ALUMNOS CON SUS CALIFICACIONES, MEDIAS Y RANGOS
E21 #CHEPARAR IMPRESORA, SI PROCEDE
E31 W+3
E41 *ALUMNO CALIFICACIONES MEDIA
E51 *RANGO
E61 CLASE,6 1 5 1 5 1 6 1 6 0*NOTAS
E71 W/D

```

Comentarios:

- línea 3.ª: es una línea para que se detenga la ejecución mientras se prepara la impresora; W es una variable que no se utilizará luego;
- líneas 4.ª y 5.ª: cabecera de la lista; línea en blanco;
- línea 6.ª: catenamos la tabla CLASE (alfanumerica) con las calificaciones en NOTAS; lógicamente hay que formatarla para pasar los números a letras; obsérvese el argumento izquierdo de formato (6 1 6 1 6 1 6 1 6 0) con 5 parejas de 6 1 (6 espacios, 1 decimal) para las 3 calificaciones y la nota media, y una última pareja (6 0: 6 espacios, 0 decimales) para la última columna, rango del alumno en la clase;
- línea 7.ª: nueva línea «de espera», antes de volver al menu principal FICHERO.

Ahora estamos en condiciones de ver cómo funciona el programa completo. Al invocar la función principal, el lector puede ir siguiendo el «diálogo» usuario-ordenador en lo que sigue:

FICHERO

O P C I O N E S

AGREGAR : 1
 CAMBIAR : 2
 CALIFICAR : 3
 ELIMINAR : 4
 LISTAR : 5
 FIN : 0

ELIJA POR SU NUMERO UNA DE LAS OPCIONES:

0:

8 ELIJA UN NUMERO DE LOS MOSTRADOS

O P C I O N E S

AGREGAR : 1
 CAMBIAR : 2
 CALIFICAR : 3
 ELIMINAR : 4
 LISTAR : 5
 FIN : 0

ELIJA POR SU NUMERO UNA DE LAS OPCIONES:

0:

1
 NUEVO ALUMNO (0=FIN)
 JORGE PERALES
 NUEVO ALUMNO (0=FIN)
 0

O P C I O N E S

AGREGAR : 1
 CAMBIAR : 2
 CALIFICAR : 3
 ELIMINAR : 4
 LISTAR : 5
 FIN : 0

ELIJA POR SU NUMERO UNA DE LAS OPCIONES:

0:

2
 LUIS MORELL

1
 MANUEL AZNAR

2
 BEATRIZ MARTINEZ

3
 MERCEDES ALMAGRO

4
 VALENTIN OLMOS

5
 ROSARIO PINILLA

6
 CARLOS CABRAL

7
 JOSE LUIS RAMOS

8
 JORGE PERALES

9
 NUMERO DE ALUMNO A CAMBIAR (0=FIN)

0:

12
 NO EXISTE ESE ALUMNO. DE NUEVO, POR FAVOR

NUMERO DE ALUMNO A CAMBIAR (0=FIN)

0:

9
 JORGE PERALES

8
 JORGE PERAL

NUMERO DE ALUMNO A CAMBIAR (0=FIN)

0:

0

O P C I O N E S

AGREGAR : 1
 CAMBIAR : 2
 CALIFICAR : 3
 ELIMINAR : 4
 LISTAR : 5
 FIN : 0

ELIJA POR SU NUMERO UNA DE LAS OPCIONES:

0:

3
 LUIS MORELL
 MANUEL AZNAR
 BEATRIZ MARTINEZ
 MERCEDES ALMAGRO
 VALENTIN OLMOS
 ROSARIO PINILLA
 CARLOS CABRAL
 JOSE LUIS RAMOS
 JORGE PERAL
 NUM ALUMNO A CALIFICAR (0=FIN)
 0

9
 JORGE PERAL

0 0 0 0 0

NUEVAS NOTAS (3) :
 3 NOTAS, POR FAVOR :
 NUEVAS NOTAS (3) :
 NUM ALUMNO A CALIFICAR (0=FIN)

O P C I O N E S

AGREGAR : 1
 CAMBIAR : 2
 CALIFICAR : 3
 ELIMINAR : 4
 LISTAR : 5
 FIN : 0

ELIJA POR SU NUMERO UNA DE LAS OPCIONES:

4
 LUIS MORELL 1
 MANUEL AZNAR 2
 BEATRIZ MARTINEZ 3
 MERCEDES ALMAGRO 4
 VALENTIN OLMOS 5
 ROSARIO PINILLA 6
 CARLOS CABRAL 7
 JOSE LUIS RAMOS 8
 JORGE PERAL 9
 NUMERO DE ALUMNO A ELIMINAR (0=FIN)

9
 TECEEE "SI" SI QUIERE ELIMINAR A JORGE PERAL
 ND
 NUMERO DE ALUMNO A ELIMINAR (0=FIN)
 0:
 7
 TECEEE "SI" SI QUIERE ELIMINAR A CARLOS CABRAL
 SI
 NUMERO DE ALUMNO A ELIMINAR (0=FIN)
 0:
 0

O P C I O N E S

AGREGAR : 1
 CAMBIAR : 2
 CALIFICAR : 3
 ELIMINAR : 4
 LISTAR : 5
 FIN : 0

ELIJA POR SU NUMERO UNA DE LAS OPCIONES:

2
 LUIS MORELL 1
 MANUEL AZNAR 2
 BEATRIZ MARTINEZ 3
 MERCEDES ALMAGRO 4
 VALENTIN OLMOS 5
 ROSARIO PINILLA 6
 JOSE LUIS RAMOS 7
 JORGE PERAL 8
 NUMERO DE ALUMNO A CAMBIAR (0=FIN)
 0:
 0

O P C I O N E S

AGREGAR : 1
 CAMBIAR : 2
 CALIFICAR : 3
 ELIMINAR : 4
 LISTAR : 5
 FIN : 0

ELIJA POR SU NUMERO UNA DE LAS OPCIONES:

5
 PREPARAR IMPRESORA, SI PROCEDE
 ALUMNO CALIFICACIONES MEDIA RANGO
 LUIS MORELL 5.0 7.0 3.0 5.1 8
 MANUEL AZNAR 7.0 5.0 6.0 6.1 4
 BEATRIZ MARTINEZ 8.0 9.0 10.0 9.1 2
 MERCEDES ALMAGRO 9.0 9.0 10.0 9.4 1
 VALENTIN OLMOS 6.0 5.0 7.0 6.1 5
 ROSARIO PINILLA 9.0 5.0 4.0 6.1 6
 JOSE LUIS RAMOS 8.0 7.0 2.0 5.7 7
 JORGE PERAL 5.0 7.0 8.0 6.7 3

O P C I O N E S

AGREGAR : 1
 CAMBIAR : 2
 CALIFICAR : 3
 ELIMINAR : 4
 LISTAR : 5
 FIN : 0

CONCLUSION

ELIJA POR SU NUMERO UNA DE LAS OPCIONES:

0:

NO OLVIDE HACER)SAVE SI HUBO CAMBIOS

)SAVE

19:55:44 86/11/22 CAP13



El capítulo 13 mostró un programa con cierta complejidad, FICHERO, que con muy pocas modificaciones podría servir (para no demasiados alumnos en no muchas clases) para un caso de aplicación real.

Ello convencerá al lector que ha seguido fiel y pacientemente el libro (sin saltarse ningún capítulo, claro) que incluso con los conocimientos ciertamente elementales que se pueden conseguir con las páginas que preceden es factible resolver problemas significativos, «obligar» el ordenador al ordenador a obedecer nuestras órdenes.

Claro es que hay varias áreas no tratadas (manejo de ficheros secundarios y de acceso directo, pantallas complejas, tratamiento de la impresora, etc.), pero ello puede ser objeto de estudios posteriores: lo visto es ya en sí útil y formativo.

Es lógico también que el lector en quien haya «prendido» la semilla APL desee practicar de verdad, no sobre el papel —por cierto muy factible cosa cuando se trabaja con APL—, sino con un ordenador real. En el apéndice B se muestra una lista, en todo caso incompleta, de máquinas que aceptan intérpretes APL.

En fin; quiere el autor ofrecer su colaboración a los lectores que tengan dudas, deseen aclaraciones u ofrezcan sugerencias; pueden escribirle a la Editorial, APARTADO 10287, 28080 MADRID.

Si el libro ha conseguido su objeto de «desmitificar» el lenguaje APL, a veces injustamente tratado de «lenguaje sólo para científicos» o de «util sólo en ciertas áreas técnicas», y de despertar el deseo de aplicarlo a problemas de índoles diversas, el autor se sentirá satisfecho de esta larga conversación con su lector.

Juan Ruiz de Torres.

Nuevo Pireo, El Palancair, Madrid
Noviembre de 1986

APENDICE A

TABLA COMPLETA DE OPERACIONES/OPERADORES APL

SÍMBOLO	DE LEFT	FUNCION MONADICA	FUNCION DIADICA
+	mas	WUELVE ARGUMENTO CON SU SIGNO	SUMA
-	menos	CAMBIA SIGNO ARGUMENTO	RESTA
*	por	CAMBIA SIGNO ARGUMENTO	MULTIPLICACION
/	entre	INVERSO	DIVIDE
1		VALOR ABSOLUTO	RESISTO
2		ELEVADO A	POTENCIACION
3		LOGARITMO NATURAL	LOGARITMO
4		PI POR	TRIGONOMETRICAS
5		FACTORIAL GAMMA	COMBINATORIA
6		no hay	COMPARACIONES
7		no hay	Y BOOLEAND
8		REONDED FOR HALF	MINIMO
9		REONDED FOR ALTO	MAXIMO
10		AZAR	AZAR SIN REPETIR
11		VECTORIZADO	CANTENAR
12		DIMENSIONES	FORMAR CUADROS
13	no	FORMATEO	FORMATEO COMPLETO
14		EDUCACION A CARL	no hay
15		no hay	TOCAR
16		no hay	DITAR
17		no hay	COMPRIMIR, EXTENDER
18		no hay	EXPANDIR, EXIENLER
19		no hay	IDEM POR FILAS
20		no hay	BRUCKER INDICE
21	100%	GENERAR SERIE NUMERICAL	PERTEINENCIA
22	spation	no existe	BASE N A BASE 10
23		no existe	BASE 10 A BASE N
24	abs y de	no existe	ASIGNAR VALOR
25	if y	no existe	no hay
26		no existe	ORDEN ALF DETERMINANTE
27		no existe	ORDEN ALF ASI ENLEVEIT
28		no existe	ROTACION
29		no existe	TRANSFORMER COMPLEJO
30		no existe	ROTACION FOR FILAS
31		no existe	SOLUCION SIST LINEAL
32		no existe	PRODUCTO INTERNO
33	dominos	no existe	PRODUCTO EXTEND

APENDICE B

APL Y LOS ORDENADORES DE DIVERSOS FABRICANTES

	Sistema operativo	Distribuido por
+ IBM PC/XT/AT	DOS	IBM; APL Informática
+ IBM S/43XX, S/30XX	MVS, VM	IBM; APL Informática
+ ATT S/3 B	UNIX	APL Informática
+ Sperry 5000	UNIX	APL Informática
+ NCR Tower	UNIX	APL Informática
+ Commodore AMIGA	APL 68000	Micro APL
+ Apple Macintosh	APL 68000	Micro APL
+ Sinclair QL	QL/APL	Micro APL
+ Atari 520, 1040	APL 6800	Micro APL
+ HP 9000	Mirage/APL	
+ DIGITAL	6800	Micro APL

Hay, seguramente, otras máquinas en que se puede trabajar con APL. La anterior lista no pretende ser exclusiva, sino informativa.

APENDICE C

DECALOGO DEL PROGRAMADOR APL

Los siguientes consejos van evidentemente dirigidos al programador novel; existen técnicas avanzadas de programación APL para los profesionales. Sin embargo, los puntos aquí expuestos pueden servir en términos generales también para ellos, y muy positivamente:

1. Describa cada función con un breve comentario; una línea bastará en la mayoría de los casos.
2. Una función no debería tener más instrucciones que líneas (a par-talla de su ordenador; así podrá examinarlas en una ojeada).
3. Escriba programas en forma estructurada; use funciones cortas para resolver cada etapa, y una para cada trabajo dentro de las distintas etapas.
4. Use funciones explícitas, cuyo resultado «alimene» a su vez a otras funciones, y haga la sintaxis cercana al lenguaje natural.
5. Los nombres de las funciones principales deberían ser lo más explícitos posible, para saber qué objeto cumplen; las funciones auxiliares deben tener nombres de tres caracteres (abreviaturas de su objetivo, si es posible); así se disminuye la ocupación de memoria y se las identifica como auxiliares.
6. Por parecidas razones, use para las variables una convención como la siguiente:
 - variables globales, externas al programa: tres letras (pueden empezar por T si son tablas/matrices, y por V si son vectores);
 - variables de transferencia, o sea, las creadas por una función para ser usadas por otra u otras: 2 letras;

— variables internas, o locales, usadas solo dentro de una función: una letra.

7. Las variables de transferencia deben hacerse locales a la función principal; las variables internas deben hacerse locales a la función que las cree y emplee.

8. Escriba y use en sus programas pequeñas funciones útiles; páselas a sus colegas; tome de ellos las ya existentes, que son muchas y excelentes; no duplique la invención de la rueda.

9. Documente su programa; escriba una explicación del funcionamiento de las funciones que crea complejas; incluya todo ello en una función DESCRIBE; al cabo de incluso muy pocos meses usted mismo agradecerá su previsión.

10. Escriba APL elegante, pero no laberíntico; tenga en cuenta que otros (y seguro que usted mismo) puedan querer usar y modificar su programa en el futuro.

Los libros siguientes son introducciones generales al APL, y se pueden obtener con facilidad:

- *Introducción al APL para el IBM Personal Computer*, por Manuel Alfonso, 1985. Editado por IBM España Distribuidora de Productos, nr. de producto SPA0200. Puede obtenerse en Concessionarios IBM.
- *Introducción al APL*. IBM España, Departamento de Educación, código 4321
- *APL: An Interactive Approach*, por Leonard Gilman & Allan Rose, J. Wiley & Sons, 3rd Edition, 1983.
- *APL: The Language and Its Usage*, por Raymond Polvka & Sandra Polkin, Prentice Hall, 1975.
- *An Introduction to APL for the IBM PC&XT*, por William H. Murray & Chris H. Pappas, Brady Communications, 1986.
- *APL/360 Programming and Applications*, por Herbert Hellerman & Ira Smith, 1976.
- *Introduction to APL2*, por John McGrew, IBM Form No. SH20-9229, 1983.
- *APL-An Introduction*, Independent Study Program, IBM Form No. SR20-7183, 1982.
- *APL Programming Guide*, IBM Form No. G320-6735, 1983.
- *APL is easy*, por Jerry R. Tuner, STSC.
- *APL-PLUS/PC*, por Jerry R. Tuner y otros, STSC.
- *APL in Practice*, por Allen Rose y Barbara Schich. John Wiley and Sons.

Revistas relacionadas con APL:

- *APL Quote Quad*, ACM, 1133 Av. of the Americas, N.Y.
- *Vector*, British APL Association, 13 Mansfield Street, London.
- *APL Market News*, Springer, POBox 503, IJmuiden, Holanda.

ENCICLOPEDIA PRACTICA DE LA INFORMÁTICA APLICADA

INDICE GENERAL

1 COMO CONSTRUIR JUEGOS DE AVENTURA

Descripción y ejemplos de las principales familias de juegos de aventura para ordenador: simuladores de combate, aventuras espaciales, búsquedas de tesoros... terminando con un programa que permite al lector construir sus propios libros de multaventura.

2 COMO DIBUJAR Y HACER GRAFICOS CON EL ORDENADOR
Desde el primer «brochazo» aprenderá a diseñar y colorear tanto figuras sencillas como las más sofisticadas creaciones que pueda llegar a imaginar, sin necesidad de profundos conocimientos informáticos ni artísticos.

3 PROGRAMACION ESTRUCTURADA EN EL LENGUAJE PASCAL

Invitación a programar en PASCAL, lenguaje de alto nivel que permite programar de forma especialmente bien estructurada, tanto para aquellos que ya han probado otros lenguajes como para los que se inician en la Informática.

4 COMO ELEGIR UNA BASE DE DATOS

Libro eminentemente práctico con numerosos cuadros y tablas, útil para poder conocer las bases de datos y elegir la que más se adecue a nuestras necesidades.

5 AÑADA PERIFERICOS A SU ORDENADOR

Breve descripción de varios periféricos que facilitan la comunicación con el ordenador personal, con algunos ejemplos de fácil construcción: ratón, lápiz óptico, marco para pantalla táctil...

6 GRAFICOS ANIMADOS CON EL ORDENADOR

En este libro las técnicas utilizadas para la animación son el resultado de unas pocas ideas básicas muy sencillas de comprender. Descubrirá los trucos y secretos de movimientos, choques, rebotes, explosiones, disparos, saltos, etc.

7 JUEGOS INTELIGENTES EN MICROORDENADORES

Los ordenadores pueden enfrentarse de forma «inteligente» ante puzzles y otros tipos de juegos. Esto es posible gracias al nuevo enfoque que ha dado la IA a la tradicional teoría de juegos.

8 PERIFERICOS INTERACTIVOS PARA SU ORDENADOR

Descripción detallada de la forma de construir, paso a paso y en su propia casa, dispositivos electrónicos que aumentarán la potencia y facilidad de uso de su ordenador: tableta digitalizadora, convertidores de señales analógicas, comunicaciones entre ordenadores.

9 COMO HACER DIBUJOS TRIDIMENSIONALES EN EL ORDENADOR

Compruebe que también con su ordenador personal puede llegar a diseñar y calcular imágenes en tres dimensiones con técnicas semejantes a las utilizadas por los profesionales del dibujo con equipos mucho más sofisticados.

10 PRACTIQUE MATEMATICAS Y ESTADISTICA CON EL ORDENADOR

En este libro se repasan los principales conceptos de las Matemáticas y la Estadística, desde un punto de vista eminentemente práctico y para su aplicación al ordenador personal. Se basan los diferentes textos en la presentación de pequeños programas (que usted podrá introducir en su ordenador personal)

11 CRIPTOGRAFIA: LA OCULTACION DE MENSAJES Y EL ORDENADOR

En este libro se presentan las técnicas de ocultación de mensajes a través de la criptografía desde los primeros tiempos hasta la actualidad, en que el uso de los computadores ha proporcionado la herramienta necesaria para llegar al desarrollo de esta ciencia.

12 APL. LENGUAJE PARA PROGRAMADORES DIFERENTES

APL es un lenguaje muy potente que proporciona gran simplicidad en el desarrollo de programas y al mismo tiempo permite programar sin necesidad de conocer todos los elementos del lenguaje. Por ello es ideal para quienes reúnan imaginación y escasa formación en Informática.

13 PRACTIQUE CIENCIAS NATURALES CON EL ORDENADOR

Ejemplos sencillos para practicar con el ordenador. Casos curiosos de la Naturaleza en forma de programas para su ordenador personal.

14 COMO SIMULAR CIRCUITOS ELECTRONICOS EN EL ORDENADOR

Introducción a los diferentes métodos que se pueden emplear para simular y analizar circuitos electrónicos, mediante la utilización de diferentes lenguajes.

15 LOS LENGUAJES DE LA INTELIGENCIA ARTIFICIAL

Libro en que se describen los lenguajes específicos para la «elaboración del saber» y los entornos de programación correspondientes. El conocimiento de estos lenguajes, además de interesante en sí mismo, es sumamente útil para entender todo lo que la Informática Artificial supondrá para el futuro de la Informática.

16 PRACTIQUE FISICA Y QUIMICA CON SU ORDENADOR

Libro eminentemente práctico para realizar pequeños «experimentos» con su ordenador y distraerse de un modo útil.

17 EL ORDENADOR Y LA LITERATURA

En este libro se examinan procesadores de textos, programas de análisis literario y una curiosa aplicación desarrollada por el autor: APOLO, un programa que compone estructuras poéticas.

18 COMO ELEJIR UNA HOJA ELECTRONICA DE CALCULO

En este título se estudian las diferentes versiones existentes de esta aplicación típica, desde el punto de vista de su utilidad para, en función de las necesidades de cada usuario y del ordenador de que dispone, poder elegir aquella que más se adecue a cada caso.

19 ECONOMIA DOMESTICA CON EL ORDENADOR PERSONAL

Breve introducción a la contabilidad de doble partida y su aplicación al hogar, con explicaciones de cómo utilizar el ordenador personal para facilitar los cálculos, mediante un programa especialmente diseñado para ello.

20 ¿MAQUINAS MAS EXPERTAS QUE LOS HOMBRES?

Después de situar los «sistemas expertos» en el contexto de la inteligencia artificial y describir su construcción, su funcionamiento, su utilidad, etc., se analiza el papel que pueden tener en el futuro (y presente, ya) de la Informática.

21 PRACTIQUE HISTORIA Y GEOGRAFIA CON SU ORDENADOR

Libro interesante para los aficionados a estas ciencias, a quienes presenta una nueva visión de cómo utilizar el microordenador en su estudio.

22 ERGONOMIA: COMUNICACION EFICIENTE HOMBRE-MAQUINA

Análisis de la comunicación entre el hombre y la máquina, y estudio de diferentes soluciones que tienden a facilitarla lo más posible.

23 EL ORDENADOR Y LA ASTRONOMIA

Los cálculos astronómicos y el conocimiento del firmamento en un libro apasionante y curioso.

24 VISION ARTIFICIAL. TRATAMIENTO DE IMAGENES POR ORDENADOR

El procesamiento de imágenes es un campo de reciente y rápido desarrollo con importantes aplicaciones en áreas tan diversas como la mejora de imágenes biomédicas, robóticas, teledelección y otras aplicaciones industriales y militares. Se presentan los principios básicos, los sistemas y las técnicas de uso más usuales.

25 LA ESTACION TERMINAL PERSONAL.

Las modernas técnicas de comunicación van permitiendo que las grandes capacidades de proceso y el acceso a bases de datos de gran tamaño estén cada día más al alcance de cada usuario (fuera ya de los Centros de Proceso de Datos).

26 EL ORDENADOR COMO MAQUINA DE ESCRIBIR INTELIGENTE

Descripción de los sistemas de tratamiento de textos existentes, análisis comparativos y estudio de posibilidades de cada uno de ellos. Guía práctica para la elección del presente paquete que más se adecue a nuestras necesidades y al ordenador personal de que dispongamos.

27 EL LENGUAJE C, PROXIMO A LA MAQUINA

Lenguaje de programación que se está imponiendo en los microordenadores más grandes, tanto por su facilidad de aprendizaje y uso, como por su enorme potencia y su adecuación a la programación estructurada. Vinculado íntimamente al sistema operativo UNIX es uno de los lenguajes de más futuro entre los que utilizan los micros personales.

28 EL ORDENADOR COMO INSTRUMENTO MUSICAL Y DE COMPOSICION

Análisis de cómo se puede utilizar el ordenador para la composición o interpretación de música. Libro eminentemente práctico, con numerosos ejemplos (que usted podrá practicar en su ordenador casero) y lleno de sugerencias para disfrutar haciendo de su ordenador un verdadero instrumento musical.

29 LA CREATIVIDAD EN EL ORDENADOR. EXPERIENCIAS EN LOGO

El LOGO es un lenguaje enormemente capacitado para la creación principalmente gráfica y en especial para los niños. En este sentido se han desarrollado numerosas experiencias. En el libro se analizan estas experiencias y las posibilidades del LOGO en este sentido, así como su aplicación a su ordenador casero para que usted mismo (o con sus hijos) pueda repetirías.

30 SISTEMAS OPERATIVOS: EL SISTEMA NERVIOSO DEL ORDENADOR

Características de diversos sistemas operativos utilizados en los ordenadores personales y caseros. Se trata de llegar al conocimiento, ameno, aunque riguroso, de la misión del sistema operativo de su ordenador, para que usted consiga sacar mayor rendimiento a su equipo.

NOTA: Ediciones Siglo Cultural, S. A., se reserva el derecho de modificar, sin previo aviso, el orden, título o contenido de cualquier volumen de la colección.