UN PROGRAMA QUE RAZONA

José María DIAZ

Complementando la entrevista que os ofrecimos en el N.º 39 con el Doctor Sierra, hemos querido mostrar esta semana una pequeña muestra práctica de un programa de inteligencia artificial, escrito en Basic.

Probablemente recordaréis lo que era un silogismo; aunque hay varios tipos de ellos, nosotros nos referiremos al siguiente:

SI A ES B
Y B ES C
ESTO IMPLICA QUE A ES C

por ejemplo, «la vida respira», «lo que respira se mueve», «luego la vida se mueve».

Nuestro programa es capaz de inferir una respuesta que no posee en memoria explícitamente, a partir de premisas lógicamente ciertas, o sea, si al programa le comunicamos las dos frases anteriores y le preguntamos ¿«la vida se mueve»?, responderá sí; este tipo de asuntos son de interés para la I.A. porque permiten simular en un ordenador, artefacto esencialmente estúpido, un proceso que las personas realizan continuamente, dar la sensación de raciocinio y de capacidad de aprendizaje; una vez que la máquina sea capaz de «deducir» «que la vida se mueve», si intentamos decirselo explícitamente, responderá que esa información puede deducirla, negándose a incorporarla en su base de datos.

El programa también es capaz de detectar si la respuesta a una pregunta es negativa según su proceso de «razonamiento», lo cual llevará a veces a respuestas sorprendentes a nuestras preguntas, e incluso falsas. Esto se desprende de las numerosas limitaciones del programa, aunque lo apasionante es, más que ver dónde acierta, ver dónde falla y porqué, permitiéndonos profundizar en la investigación y mejorarlo paso a paso.

Aclarar conceptos

Antes de describirlo, tal vez nos ayude a aclarar ideas y a comprender porqué Microsherlock tiene graves limitaciones, echar un vistazo a una serie de conceptos esenciales para que una máquina «razone».

Todos los lenguajes que se pueden emplear en I.A., como el clásico LISP y las versiones actuales de LOGO implementadas en microordenadores, poseen algo en común que simplifica enormemente este tipo de programas capaces de inferir respuestas: el procesamiento de listas de propiedades.

Vamos a tratar de explicarlo un poco: por listas se entiende una secuencia de expresiones simbólicas, esto es, «Luis es rubio» puede tratarse como una lista, y para nuestros propósitos, la consideraremos dividida en tres partes, un identificador, «Luis», una propiedad, «es», y un valor, «rubio».

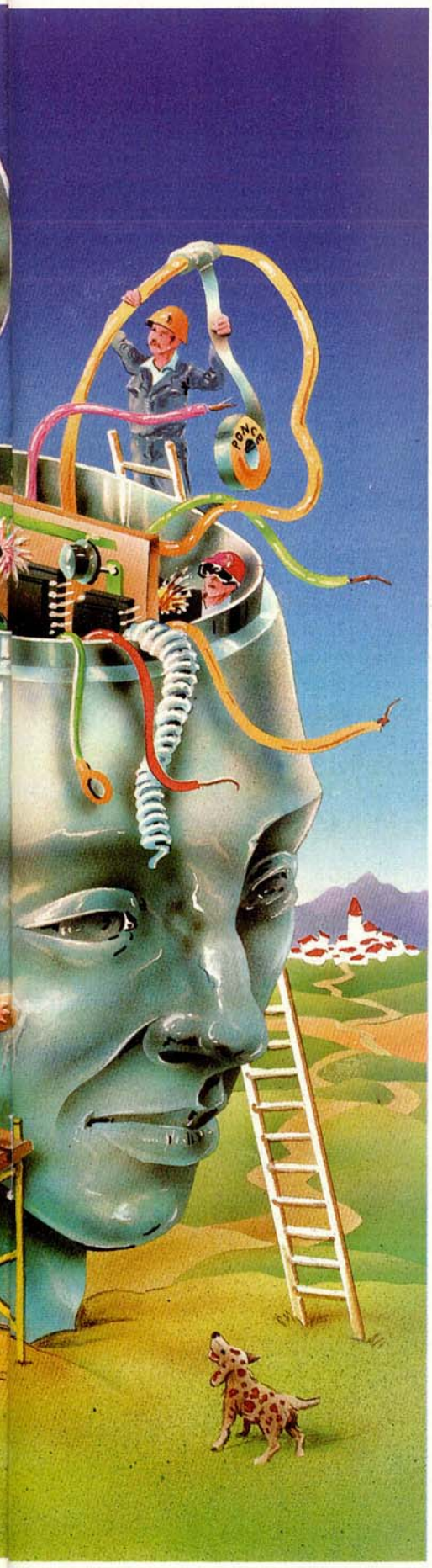
Así, podemos decir que el identificador «Luis» posee el valor «rubio» bajo la propiedad «es», y, por lo tanto, la lista de propiedades de «Luis» bajo «es» es «rubio». Si decimos a continuación «Luis es ingeniero», estos lenguajes tienen una serie de instrucciones que nos permiten añadir el nuevo valor a su lista de propiedades con lo cual ésta quedaría como «ingeniero rubio». Por último, la afirmación «Luis tiene dos brazos», crearía una nueva lista de propiedades para «Luis» bajo la propiedad «tiene» y así hasta el infinito. Todo esto es hecho automáticamente por el propio lenguaje y nosotros, en cualquier momento, podemos manipular e inspeccionar estas listas de propiedades a nuestro antojo.

Supongamos que comunicamos a nuestro hipotético programa escrito en LISP o LOGO, las siguientes aserciones:

1. Luis es ingeniero.
2. Un ingeniero es una persona.
3. Una persona tiene piernas.

y le preguntamos ¿«Luis tiene piernas?

Nuestro programa teórico haría lo siguiente: miraría la lista de propiedades de «Luis» bajo la propiedad «tiene», y no obtendría la lista «piernas»; antes de darse por vencido, miraría si existe algo sobre «Luis» bajo la propiedad «es», obtendría «ingeniero» y repetiría el proceso anterior, pero ahora la pregun-



ta concerniría a la lista ¿«ingeniero tiene piernas»?; tampoco obtendríamos lo que estamos buscando, así que se volvería a repetir el proceso, pero ahora de la lista de propiedades de «ingeniero» bajo «es» saldría «persona», con lo que la nueva pregunta ¿«persona tiene piernas»? daría como respuesta sí, ya que «piernas» está en la lista de propiedades de «persona» bajo la propiedad «tiene». Como nuestros lectores habrán podido observar, este método es completamente recursivo, es decir, el procedimiento que escribiríamos tendría dos argumentos que se modificarían en el mismo procedimiento, antes de volver a llamarse a sí mismo con los nuevos argumentos, facilitando enormemente la escritura de este tipo de programas.

La recursividad

Llegamos aquí a la segunda condición que un lenguaje I.A. debe tener, la recursividad.

De momento, el programa es bastante triste, ya que el Basic del Spectrum no posee ni listas de propiedades, ni recursividad, ni nada parecido. Este es el motivo de que Microsherlock tenga una serie de limitaciones, que no se pueden eliminar si queremos que el programa no sea ni demasiado grande ni demasiado complejo.

Microsherlock, cuando arranca, presenta un pequeño menú de tres opciones:

1. Aprende.
2. Contesta.
3. Muestra.

la primera nos servirá para suministrar información al programa, la segunda para contestar a nuestras preguntas, y la tercera nos enseñará lo que tiene almacenado en la base de datos acerca de un determinado sujeto.

En aras de la sencillez, nos hemos saltado a la torera (mea culpa) el escribir un analizador de lenguaje (nosotros y todo el mundo, porque hasta ahora nadie lo ha conseguido completamente), con idea de aumentar la capacidad de respuesta del programa; así, se requiere un pequeño esfuerzo de imaginación por parte del usuario; si queremos comunicar al programa que «un hombre siente», al escoger la opción 1 aparecerá la pregunta ¿«sujeto»?, y le daremos «hombre», después aparecerá ¿«valor»? y le diremos «siente». Lo único que hacemos es entregar al programa el sujeto y el valor directamente, cosa que haría el analizador de lenguaje al acabar su proceso. Lo mismo ocurre con la opción 2. Veremos que, aunque mantengamos la estructura total de la pregunta en la mente, si la información se le da al programa encadenada lógicamente, responde con corrección a casi cualquier propiedad como «es», «tiene», «tendría», etc. Esto se descubrirá sobre la marcha con la experimentación, teniendo en cuenta que si damos datos inconexos o al azar al programa, obtendremos respuestas de lo más divertido.

	1	2	3	4	5
1	HOMBRE	SIENTE	MUEVE	RESPIRA	DESPLAZA
2	SIENTE	HOMBRE	SIENTE	SIENTE	RESPIRA
3		MUEVE		DESPLAZA	
4		RESPIRA			
5					

DIMENSION VERTICAL ▽

DIMENSION HORIZONTAL ▷

Situación de la matriz de datos (N$) tras introducir la frase «lo que respira se desplaza.

Para imitar en lo posible las listas de propiedades, hemos recurrido a una estructura de datos muy conocida, de la que pueden encontrarse referencias en cualquier libro: una tabla de referencias cruzadas, representada en Microsherlock por una matriz bidimensional de 25 x 25 (la tercera dimensión es la que indica la longitud máxima de cada elemento de la matriz).

El programa está estructurado como un bucle que se encarga de manejar tres subrutinas principales (ver diagrama 1), algunas de las cuales a su vez, llaman a otras de la siguiente manera:

a) Rutina organizar datos (línea 250) accedemos a ella al elegir la opción aprende del menú (Diagrama 2).

Primero se llama a la subrutina INPUT (línea 170) para obtener la frase. A continuación, se comprueba si el sujeto existe y si hay sitio para almacenarlo (línea 300). Si no hay sitio, se produce un mensaje de error, y si lo hay, tiene lugar una nueva bifurcación dependiendo de que el sujeto existiera antes o no (línea 330). Si no existe, la información es nueva y debe ser añadida a la base de datos; de ellos se encarga la rutina Nueva información (línea 380). Si el sujeto ya existía, el control lo toma la rutina Información adicional (línea 460), la cual comprueba primero que la información ni pueda inferirse ni sea falsa (línea 480); una vez solucionado este pequeño trámite, coloca el sujeto y el valor en los lugares apropiados (línea 520, 550 y 560).

b) Rutina mostrar datos (línea 1090) no merece mayor comentario. Simplemente busca el sujeto y, si lo encuentra, lista la información disponible acerca de él.

c) Rutina Inferir respuesta (línea 600) esta subrutina es la clave del programa,y consta de la principal y dos secundarias que sirven, una, para simular algo parecido a la recursividad, la rutina REINTENTAR (línea 880), y la otra, para tomar en cuenta hasta un cierto punto el pedir información incompleta, rutina MATIZA (línea 980). Lo primero que hace es buscar el valor (líneas 650-670) y el sujeto (líneas 690-710). Si los dos o uno de los dos no existen, imprime el mensaje correspondiente y retorna al bucle principal (líneas 720-730).

A continuación, el programa investiga lo que tiene almacenado bajo el sujeto, para cubrir la posibilidad de que le preguntemos algo que le hemos dicho explícitamente (por ejemplo, si le decimos «un hombre siente» y le preguntamos ¿«un hombre siente»?); si lo encuentra, deduce que la respuesta es sí y retorna (línea 750).

Por fin, el programa buscará el valor mirando si está relacionado con el sujeto; si lo está (línea 790) llamará a la rutina REINTENTAR y el proceso se repetirá con toda la información almacenada debajo del sujeto hasta que ésta se agote, en cuyo caso la respuesta sería no (línea 920), retornando al bucle principal. Si la respuesta es afirmativa, el programa lo detecta en la línea 810 y retorna con el flag «si» puesto a uno.

Realización práctica

Vamos a ver todo esto paso a paso con un ejemplo cuya situación final se refleja en el cuadro 1.

Arrancamos el programa y escogemos la opción 1, aprende. A la pregunta de ¿sujeto? respondemos «hombre» y la de valor «siente»; se ejecutará NUEVA INFORMACION y Microsherlock colocará el sujeto y el valor como se muestra en el cuadro 1, e inmediatamente a continuación, los colocará invertidos.

```
   10 REM PROGRAMA INFER
   20 REM ** INICIALIZACION **
   30 GO SUB 1240
   40:
   50 REM ** BUCLE PRINCIPAL **
   60 CLS
   70 LET veces=veces+uno
   80 PRINT AT VAL "10",VAL "8";"
1-APRENDE"
   90 PRINT AT VAL "12",VAL "8";"
2-CONTESTA"
  100 PRINT AT VAL "14",VAL "6";"
3-MUESTRA"
  110 PAUSE 0: IF INKEY$<"1" OR I
NKEY$>"3" THEN GO TO 110
  120 GO SUB (250 AND INKEY$="1")
+(600 AND INKEY$="2")+(1090 AND
INKEY$="3")
  130 IF si THEN PRINT "> LA RESP
UESTA ES SI": LET si=cero
  140 PRINT "> PULSA UNA TECLA":
PAUSE 0: GO TO 60
  150 REM FIN BUCLE PRINCIPAL
  160:
  170 REM ** RUTINA INPUT **
  180 INPUT "SUJETO..."; LINE s$
  190 INPUT "VALOR..."; LINE v$
  200 LET longs=LEN s$: LET longv
=LEN v$
  210 IF (longs>long) OR (longv>l
ong) THEN PRINT "> TU FRASE ES D
EMASIADO LARGA": GO TO 170
  220 RETURN
  230 REM ** FIN RUTINA INPUT **
  240:
  250 REM ** ORGANIZA DATOS **
  260 CLS : PRINT AT VAL "10",VAL
 "5";"> ORGANIZANDO LA TABLA": P
RINT AT VAL "12",VAL "5";("> EJE
CUTANDO RUTINA 250" AND veces<nv
eces)
  270 GO SUB 170
  280 LET conta=cero: LET flag=ce
ro
  290 LET conta=conta+uno
  300 LET flag=(N$(uno,conta,uno)
=" ")+(N$(uno,conta, TO longs)=s
$)
  310 IF (flag=cero) AND (conta<h
dim) THEN GO TO 290
  320 IF conta>=hdim THEN PRINT "
> NO QUEDA SITIO": RETURN
  330 IF N$(uno,conta,uno)=" " TH
EN GO SUB 380: RETURN
  340 GO SUB 460
  350 RETURN
  360 REM * FIN ORGANIZA DATOS *
  370:
  380 REM * NUEVA INFORMACION *
  390 PRINT ("> EJECUTANDO RUTINA
  380" AND veces<nveces)
  400 PRINT "> NUEVA INFORMACION"
  410 LET N$(uno,conta)=s$: LET N
$(dos,conta)=v$
  420 LET N$(uno,conta+uno)=v$: L
ET N$(dos,conta+uno)=s$
  430 PRINT "> ENTIENDO": RETURN
  440 REM FIN NUEVA INFORMACION
  450:
  460 REM INFORMACION ADICIONAL
  470 LET temp=uno: PRINT "> INFO
RMACION ADICIONAL": PRINT ("> EJ
ECUTANDO SUBRUTINA 460" AND vece
s<nveces)
  480 GO SUB 630: IF respuesta=ce
ro THEN PRINT "> DEDUZCO QUE ESA
 INFORMACION ES FALSA": PRINT ">
 NO INCORPORADA": RETURN
  490 IF si THEN LET si=cero: PRI
NT "> PUEDO DEDUCIR ESA INFORMAC
ION": PRINT ("> POR TANTO, ES RE
DUNDANTE" AND veces<nveces): PRI
NT "> NO INCORPORADA": RETURN
  500 LET temp=temp+uno
  510 IF N$(temp,conta,uno)<>" "
THEN GO TO 500
  520 LET N$(temp,conta)=v$
  530 LET conta=conta+uno
  540 IF N$(uno,conta,uno)<>" " T
HEN GO TO 530
  550 LET N$(uno,conta)=v$
  560 LET N$(dos,conta)=s$
  570 PRINT "> INCORPORADA": PRIN
T "> AHORA YA LO SE": RETURN
  580 REM FIN ADICIONAL
  590:
  600 REM * INFERIR RESPUESTA *
  610 CLS : PRINT AT VAL "10",VAL
 "5";"> INFIRIENDO RESPUESTA": P
RINT : PRINT ("> EJECUTANDO RUTI
NA 600" AND veces<nveces)
  620 GO SUB 170
  630 LET vez=uno: LET repite=cer
o: LET respuesta=uno: LET tempv=
cero
  640 PRINT "> ESTUDIANDO...";s$
  650 FOR I=uno TO hdim
  660 IF N$(uno,I, TO longv)=v$ T
HEN GO SUB 990
  670 NEXT I
  680 LET temps=cero: LET lugar=u
no: LET si=cero: LET t$="": LET
longs=LEN s$
  690 FOR I=uno TO hdim
  700 IF N$(uno,I, TO longs)=s$ T
HEN LET temps=I: LET I=hdim: IF
vez=uno THEN LET repite=temps
  710 NEXT I
  720 IF (temps=cero) AND (tempv=
cero) THEN PRINT "> NO TENGO DAT
OS PARA RESPONDER": RETURN
  730 IF (temps=cero) OR (tempv=c
ero) THEN PRINT "> NO LO SE": RE
TURN
  740 FOR I=dos TO vdim
  750 IF N$(I,temps, TO longv)=v$
 THEN LET si=uno: LET I=vdim: RE
TURN
  760 IF N$(I,temps,uno)=" " THEN
 LET si=cero: LET I=vdim
  770 NEXT I
  780 LET t$=N$(lugar,temps, TO l
ongs)
  790 IF t$(uno)=" " THEN GO TO 8
90
  800 FOR I=uno TO vdim
  810 IF N$(I,tempv, TO longs)=t$
 THEN LET si=uno: LET I=vdim: RE
TURN
  820 NEXT I
  830 LET lugar=lugar+uno
  840 GO TO 780
  850 RETURN
  860 REM FIN INFERIR RESPUESTA
  870:
  880 REM REINTENTAR
  890 PRINT ("> EJECUTANDO RUTINA
  880" AND veces<nveces)
  900 PRINT "> LO INTENTO DE NUEV
O"
  910 LET vez=vez+uno
  920 IF N$(vez,repite,uno)=" " T
HEN PRINT "> LA RESPUESTA ES NO"
: LET respuesta=cero: RETURN
  930 LET s$=N$(vez,repite)
  940 PRINT "> ESTUDIANDO...";s$
  950 GO TO 680
  960 REM FIN REINTENTAR
  970:
  980 REM MATIZA
  990 PRINT ("> EJECUTANDO RUTINA
  980" AND veces<nveces)
1000 IF longv=long THEN GO TO 10
60
1010 IF N$(uno,I,longv+uno)=" "
THEN GO TO 1060
1020 PRINT "> POR ";v$: PRINT ">
 ENTIENDO QUE TE REFIERES A": PR
INT N$(uno,I): PRINT "> ES CIERT
O? (S/N)": PAUSE 0
1030 IF INKEY$="N" THEN RETURN
1040 PRINT "> PERFECTO"
1050 LET v$=N$(uno,I): LET longv
=LEN v$
1060 LET tempv=I: LET I=hdim
1070 RETURN
1080 REM FIN MATIZA
1090 REM * MOSTRAR DATOS *
1100 CLS : PRINT ("> EJECUTANDO
RUTINA 1090" AND veces<nveces):
PRINT
1110 PRINT "> DIME EL SUJETO": P
RINT "> QUE QUIERES EXAMINAR": P
RINT "> TECLEA 'FIN' CUANDO TERM
INES"
1120 LET c=cero
1130 INPUT "QUIERES VER..."; LIN
E s$
1140 IF s$="FIN" THEN PRINT "> D
E ACUERDO": LET s$="": RETURN
1150 LET c=c+uno
1160 IF (c<hdim) AND (N$(uno,c,
TO LEN s$)<>s$) THEN GO TO 1150
1170 FOR I=uno TO vdim
1180 PRINT N$(I,c)
1190 IF N$(I,c,uno)=" " THEN LET
 I=vdim
1200 NEXT I
1210 GO TO 1120
1220 REM * FIN MOSTRAR DATOS *
1230:
1240 REM ** INICIALIZACION **
1250 CLS : PRINT AT VAL "0",VAL
"5";"© MICROHOBBY SEMANAL"
1260 PRINT AT VAL "8",VAL "5";"
 ESTOY INICIALIZANDOME": PRINT A
T VAL "10",VAL "5";"> EJECUTANDO
 RUTINA 1240"
1270 LET hdim= VAL "25": LET vdi
m=hdim: LET long=VAL "30"
1280 DIM N$(vdim,hdim,long)
1290 LET cero=VAL "0": LET uno=V
AL "1": LET dos=VAL "2": LET tem
p=uno: LET si=cero: LET veces=ce
ro
1300 LET longs=uno: LET longv=un
o: LET nveces=VAL "10"
1310 POKE VAL "23658",VAL "8"
1320 PRINT AT VAL "12",VAL "5";"
> INICIALIZACION CONCLUIDA";AT V
AL "14",VAL "4";"> BIENVENIDO A
MICROSHERLOCK"
1330 PAUSE 100: CLS : RETURN
```

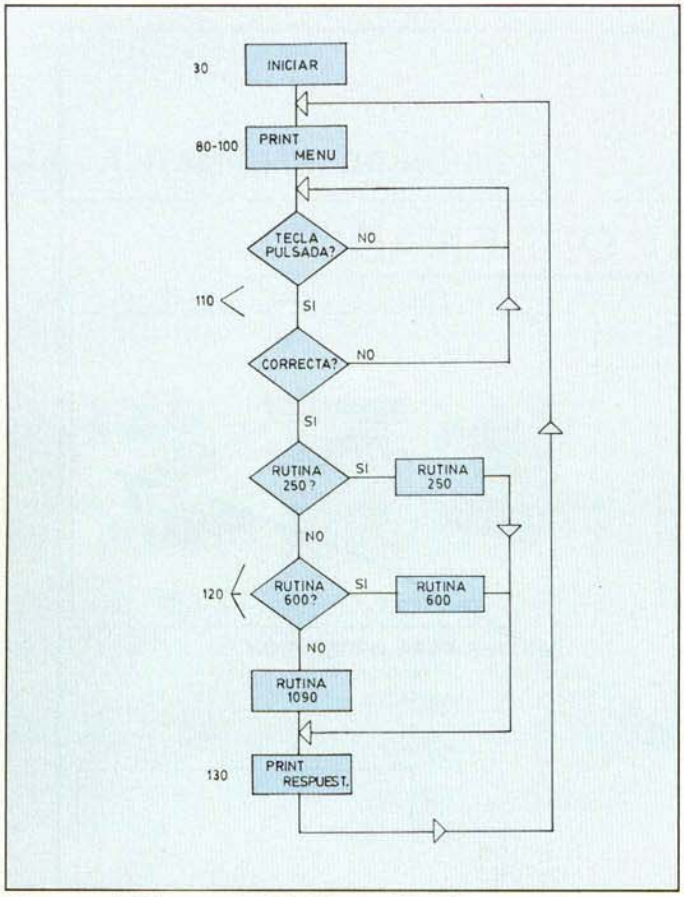


Diagrama 1. Bloque principal. Líneas 1Ø-15Ø.

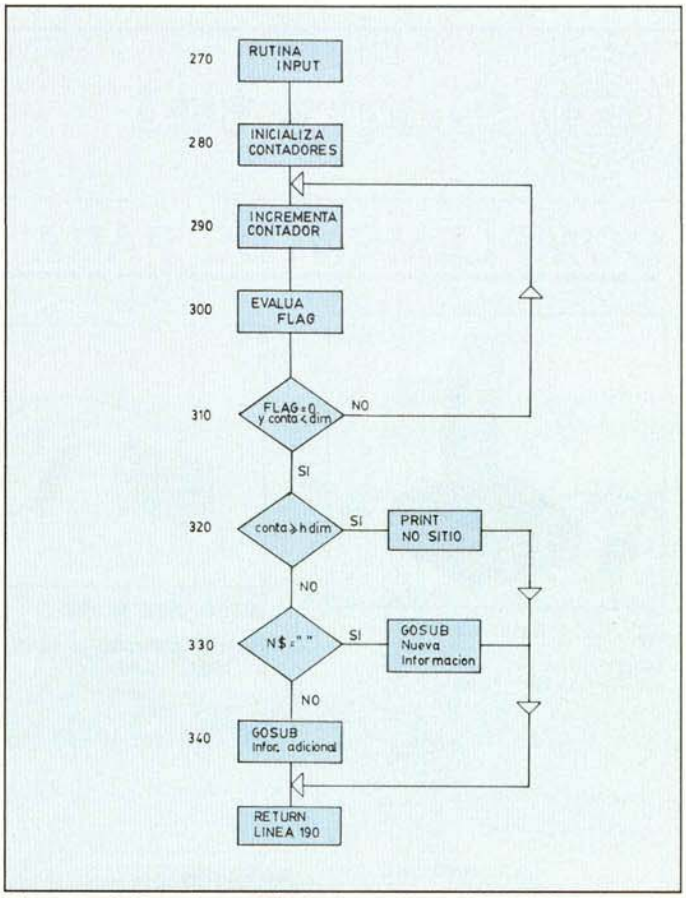


Diagrama 2. Organizar tabla de datos. Líneas 25Ø-36Ø.

De nuevo en aprende, decimos algo así como «lo que siente se mueve» dando «siente» como sujeto y «mueve» como valor; se ejecutará INFORMACION ADICIONAL, el programa buscará «siente» y debajo, donde encuentre hueco, colocará «mueve» y acto seguido lo invertirá en los dos elementos siguientes de la matriz; nuestra siguiente frase imaginaria sería «lo que siente respira», se repetiría el proceso anterior, y por último «lo que respira se desplaza» llegando a la situación del cuadro número 1.

Ahora escogemos la opción 2, contesta, y preguntamos ¿«el hombre siente»? (caso 1), dando «hombre» como sujeto y «siente» como valor.

Microsherlock buscará «hombre», encontrándolo en el elemento 1,1 de la matriz, luego buscará «siente», que está en 1,2 y mirará debajo de «hombre» para ver si encuentra «siente»; como es así en este caso, no buscará más y responderá sí.

Nuestra siguiente pregunta es un poco más compleja: ¿«el hombre respira»? (caso 2), dando como siempre «hombre» como sujeto y «respira» como valor. El programa, como antes, buscaría «hombre» y «respira»; debajo de «hombre» no existe «respira», así que el programa vuelve a «respira» (elemento 1,5) y comparará las categorías existentes debajo de «hombre» y «respira» para ver si alguna coincide; la palabra «siente» existe debajo de ambos, por tanto, el programa responde sí.

Hasta este momento, la rutina REINTENTAR no ha entrado en servicio, pero si preguntamos ¿«el hombre se desplaza»? es fácil ver que los dos casos anteriores fallan; entonces, el programa transfiere control a REINTENTAR ya que debajo de «hombre» hay otra categoría «siente», repite el caso dos y descubre que la respuesta es sí, porque ahora la pregunta hace referencia a «siente» y «desplaza», como cuando hablabamos del LISP y el LOGO.

Sólo nos queda la rutina MATIZA. Imaginemos que en el cuadro 1, en lugar de «respira» pusiera «respira deprisa», e hiciéramos las mismas preguntas; Microsherlock, mediante esta rutina, se daría cuenta de que «respira» es una parte de «respira deprisa», nos pediría confirmación y respondería como antes, sí.

En fin, sólo nos queda decir que este programa admite multitud de mejoras, omitidas aquí porque harían aumentar su complejidad y su longitud. Las principales que se nos ocurren afectan a las rutinas MATIZA y REINTENTAR; puede intentarse que la primera detectará que «deprisa» también es parte de «respira deprisa» y que la segunda fuera un poco más recursiva, en el sentido de mirar no sólo «hombre» y «siente» como en el ejemplo, sino también todas las categorías debajo de «siente»; ahora bien, esto demoraría el programa notablemente.

Otra forma de aumentar la eficiencia, es emplear una matriz de tres dimensiones, en la cual la tercera se usará para guardar una propiedad diferente; así, «Luis es ingeniero» y «Luis tiene piernas» se guardarían en los lugares reservados para las propiedades «es» y «tiene» respectivamente.

El lector obsevará que hay algunas partes del programa que son algo redundantes, sobre todo en lo relativo al flags, y que cada subrutina tiene sus propias variables definidas al comenzar en lugar de ponerlas todas juntas en la rutina de inicialización.

Esto se ha hecho así, a pesar de que tal vez relentize algo el programa, pensando en la facilidad de comprensión y por imitar al máximo una programación estructurada en «procedimientos», como de hecho habría ocurrido de utilizar un lenguaje más apropiado.