FORTH: POTENCIA Y VELOCIDAD EN ALTO NIVEL

David Sopuerta

Como todos sabemos, el lenguaje utilizado por excelencia para programar nuestros ordenadores caseros ha sido el Basic. Pero los aficionados inquietos, se habrán dado cuenta rápidamente que no es el lenguaje de programación ideal para que nuestro Amstrad actúe de una manera eficaz en todas las ocasiones y, que algunas aplicaciones como, por ejemplo, los juegos de arcade y similares, están pidiendo a gritos que utilicemos un sistema más rápido y compacto a la hora de codificarlos.

La primera solución que se nos ocurre es escribir este tipo de programas en código máquina. ¡Qué problema! A la mayoría de nosotros puede que hacerlo nos resulte todavía bastante difícil y penoso ya que se trata de un lenguaje en el que hay que **«hilar»** muy fino, teniendo muy claro qué es lo que queremos hacer y conociendo con precisión las instrucciones internas del micro.

Además, los programas generados son largos y de difícil seguimiento en caso de error. Tiene un gran número de instrucciones elementales entre las que nos resultará relativamente fácil perdernos. Así que de momento...

Un camino más fácil y entretenido es utilizar alguno de los muchos lenguajes de alto nivel que ya están a disposición de nuevo **Amstrad.** Entre ellos están incluidos el Pascal, el Forth y el Logo y, si su ordenador está equipado con una unidad de disco, también podrá hacer sus programas en Lisp, Prolog, Fortran, C y muchos más.

Cada uno de ellos tiene sus ventajas e inconvenientes en las diferentes aplicaciones que les demos. Mientras un lenguaje puede parecer ideal para una cosa en concreto, a lo mejor resulta ser demasiado lento o necesitar demasiada memoria en otras ocasiones.

En pocas palabras. Lo que realmente estamos necesitando es un lenguaje «ideal» que, en conjunto, sea suficientemente rápido para cualquier requisito, que no «despilfarre» demasiada memoria y que, por supuesto, sea relativamente fácil de aprender y utilizar.

Forth: historia de la eficacia

El lenguaje que, en principio, se adapta mejor a estos requisitos es el Forth y, no debe sorprenderle que sea el segundo en popularidad entre los usuarios de ordenadores caseros. Es sencillo, compacto, pensado para aplicaciones de uso general, ideal para emplearlo en multitud de problemas y su aprendizaje no entraña ninguna dificultad, a pesar de su estructura y vocabulario poco usuales.

El Forth nació alrededor de 1969 y en un principio se utilizó para controlar los complejos movimientos de los grandes telescopios. Desde entonces ha tenido una amplia y variada gama de usuarios para una no menos extensa galería de aplicaciones.

Posee alguna de las características avanzadas de los lenguajes de alto nivel, tales como estructuras de bucles y análisis de condiciones muy complejas, y genera programas muy uniformes que se ejecutan a una gran velocidad, aproximadamente 10 veces más deprisa que el Basic. A esto hay que añadir que podemos modificar y ampliar el lenguaje dotándolo de nuevas palabras «clave» a medida que vayan surgiendo a lo largo de cualquier aplicación.

Nuestra primera tarea, antes de comenzar a desentrañar las maravillas de este nuevo lenguaje, es, por supuesto, teclear el programa que nos servirá para ejecutar en el **Amstrad** las instrucciones que escribamos en Forth.

Un intérprete para aprender

Dese cuenta de algo importante. Este programa no nos genera realmente una **versión del Forth**, simplemente simula esta operación. Su misión es convertir cada nueva palabra en un código que responda a un formato interno especial que se corresponde con unas instrucciones Basic que son las que realmente realizan el trabajo.

Y como a continuación el **Amstrad** ha de interpretar estas sentencias Basic, no se extrañe que en esta ocasión cualquier programa creado en Forth se ejecutará bastante lentamente. Pero no se desespere. Por lo demás esta versión nos permite utilizar un Forth bastante parecido al que se usa realmente. A pesar de su lentitud, nos permitirá hacer experimentos con este potente lenguaje empleando las técnicas sugeridas en este artículo. Cuando su aprendizaje haya terminado ya estaremos capacitados para decidir si Forth es **«nuestro»** lenguaje. De ser así, ya sabe lo que le tocará hacer: adquirir una de las versiones comerciales disponibles para el **Amstrad.**

Al principio puede resultarnos un poco incómodo de utilizar debido principalmente a que emplea una notación diferente a la que nosotros estamos acostumbrados. En vez de escribir instrucciones en la forma que lo hace el Basic, todas las sentencias Forth, o **«palabras»** como de ahora en adelante las conoceremos, necesitan tener sus argumentos —o números con los que trabajan— **«delante»** de cada comando y no después como es el caso de la mayor parte de los lenguajes.

El Forth: es un lenguaje basado en el concepto de stack

Por ejemplo, si queremos sumar dos números y después imprimir su resultado, lo conseguiremos con la siguiente instrucción:

```
PRINT 3+8
```

Pero en Forth no se haría así. Tendremos que escribir algo parecido a:

```
3 8+.
```

donde el punto (.) es la «palabra» Forth empleada para visualizar en la pantalla. Esta forma de escribir las instrucciones es lo que se conoce como «Notación Polaca Inversa» y le aseguramos que no es tan enrevesada como parece, así que no se sienta desanimado por esta novedad.

Sin embargo, no es la única diferencia existente. El Forth trabaja empleando una parte de la memoria llamada «stack» o pila para guardar los valores que en un determinado momento van a tener los argumentos que vayamos a pasar a una **«palabra».**

Su funcionamiento es muy sencillo. Imagine que está recogiendo platos y **«apilándolos»** uno encima del otro. Es evidente que el último plato que hayamos tomado será el que coloquemos en la parte superior de la **«pila».** Una vez colocados, necesitamos coger uno. *¿Cuál será?*

A menos que sea un malabarista e intente demostrarlo sacando uno de los platos de abajo, lo más normal y lógico es que el elegido sea precisamente el que está colocado encima de todos. Así sucede con el «stack», el último dato que hayamos colocado en él es generalmente el primero que después vamos a sacar de allí.

Es lo que los técnicos han bautizado como memoria **«LIFO»** (o **«last input-first output»**) que quiere decir: el último elemento que entra es el primero que sale.

La manera en la que el Forth interpreta las instrucciones, o **«palabras»,** que escribimos anteriormente es bastante sencilla. Coloca en el stack los números que vamos a sumar en el orden que se los hemos dado —primero el 3 y encima el 8. A continuación se ejecuta la palabra Forth +.

El modo de operación es muy semejante en la mayor parte de las sentencias Forth: se sacan los dos números del stack, se procesan en la forma oportuna y el resultado se vuelve a guardar en el stack para que pueda ser manejado por sucesivas palabras.

En este caso, se cogen los dos datos que ocupan las posiciones más altas de la pila, se suman y se devuelve este dato al stack.

A continuación la siguiente palabra (.) extrae el número que ahora esté colocado encima (el resultado de la operación anterior) y lo saca en la pantalla. El Forth además imprime el mensaje de **«OK»** para convencernos de que la instrucción se ha ejecutado sin errores. Observe y tenga siempre en cuenta que tanto los datos como las palabras Forth han de estar separadas por un espacio en blanco. ¡No lo olvide!

Estos comandos dejan el stack exactamente igual que estaba antes de ejecutarlos. Por nuestra parte le damos un consejo: es siempre conveniente que así ocurra ya que de esta forma permitimos a las **«palabras»** siguientes, operar con los valores que se colocaron en la pila antes de ejecutarse la anterior secuencia de instrucciones.

El límite superior del stack siempre contiene el útlimo número introducido y si metemos en él un nuevo valor, el anterior queda debajo de modo que el elemento que hemos almacenado más recientemente es el que ocupará la parte más alta de la pila, de ahí que se le llame **«top of stack»** (superior de la pila).

Quizá nos llame la atención, por lo que hemos visto hasta ahora, que el Forth sólo sea capaz de sumar dos números y no tres, cuatro o todos los que queramos. Pero es lógico, la «palabra« Forth «+» opera siempre solamente con dos números (al fin y al cabo es lo que nosotros hacemos con cada una de sus ci-

fras) y a continuación devuelve al stack un único resultado. Y esta forma de trabajo no sólo es válida para la suma sino que podemos extenderlo al resto de las operaciones aritméticas empleadas en Forth, tales como multiplicaciones y divisiones.

Pero este modo de desarrollar el lenguaje no nos impide utilizar expresiones más complejas. Solamente será necesario poner un poco más cuidado en lo que estamos haciendo a la hora de decidir cómo organizarlas según esta nueva notación.

Evaluación de expresiones en Forth

Pongamos un ejemplo. Si queremos evaluar en Forth la expresión:

15+2×9

debemos comenzar multiplicando 2 por 9 (recuerde la prioridad de operaciones) para encontrar un resultado intermedio que sumado luego a 15, nos dé el valor final de la expresión.

Otra forma de calcularlo sería sumando primero 15 y 2 y después multiplicar el resultado por 9. Observe que la solución obtenida es completamente diferente de la anterior. ¿Cuál es la buena?

Recuerde que en Basic existe un orden de ejecución en las operaciones matemáticas y que el producto es más prioritario que la suma. Por tanto, el Basic y la mayoría de los lenguajes utilizarán el primer método de evaluación.

Sigámosle también en Forth. Para multiplicar 2 y 9 tendremos que teclear:

```
2 9 *
```

y nos dejará el resultado (18) en el elemento superior del stack. Podemos comprobarlo visualizándole en la pantalla usando la palabra **«.»** pero tenga en cuenta que, si así lo hacemos desaparecerá de la pila. De manera que como vamos a necesitar este valor en la siguiente parte del cálculo es mejor que lo dejemos donde estaba.

Después necesitamos sumar 15 al elemento superior de la pila. Pues teclee:

```
15+
```

y colocará el resultado en el nuevo «TOS» (top of stack) pudiéndolo imprimir a continuación. La primitiva expresión Basic se ha convertido en:

```
2 9*15+.
```

que es el comando Forth que nos va a sacar en la pantalla el resultado correcto de la evaluación: 33.

Sobre los cálculos matemáticos queremos puntualizar algo. La mayor parte de las versiones del Forth tiene unos operadores aritméticos que utilizan solamente números enteros y además dan resultados que son también enteros.

Escriba ahora:

```
9 4/.
```

que tiene toda la pinta de ser el equivalente en Forth a:

```
PRINT 9/4
```

y la respuesta obtenida es 2 en lugar de 2.25 que sería el valor exacto. De esta forma confirmamos que el número que nos devuelve la operación es también entero. Así que de momento queda bastante claro que no podemos usar números con punto decimal en esta versión del lenguaje.

Pero también está restringido el rango de los números. Intente teclear:

```
1000 200 *.
```

Parece que la respuesta que aparece en la pantalla no está muy de acuerdo con lo que esperábamos. La aritmética en Forth está pensada para ser utilizada en micros, del mismo modo que el lenguaje. Por tanto, trabajará con números que puedan ser representados con 16 bits *(o dígitos binarios)* o lo que es lo mismo, con valores comprendidos dentro del rango de —32768 a 32767. De ahí el mensaje, aunque sea ficticio, de **«stack lleno»:** la pila no está llena, lo que se produce es un **«overflow»** o desbordamiento de la capacidad numérica de su micro.

Esta notación aritmética no es tan compleja como parece. El mejor camino para utilizarla con soltura es intentar emplearla muchas veces para hacer en Forth todas las operaciones que se nos ocurran. Después de realizar unas cuantas prácticas verá que su uso es casi tan sencillo como la empleada en la aritmética habitual con una ventaja: es mucho más potente.

La potencia del Forth está en crear nuevas órdenes en Forth

Sin embargo, donde radica el poder real del lenguaje Forth es en el hecho de poder definir nuevas «palabras» que se añaden inmediatamente a su vocabulario básico así como definir con otro nombre las ya existentes.

La **«palabra»** Forth es algo equivalente a las subrutinas, o partes de un programa Basic a las que se salta tras un GOSUB y siempre terminan con RETURN.

De un modo sencillo, podemos decir que una **«palabra»** es una serie de instrucciones, y sus correspondientes parámetros, que están agrupadas bajo un nombre que incorporamos, tras definirla, al vocabulario Forth.

Para ejecutar esta secuencia basta con invocarla, o llamarla, por su nombre y el programa salta a la serie de instrucciones, así bautizadas, y las ejecuta.

Supongamos que preferimos utilizar palabras castellanas como operadores aritméticos en lugar de los signos matemáticos y también

emplear ESCRIBIR en lugar del punto Forth. Todo lo que hay que hacer es definir las nuevas palabras.

Para ello escribimos:

```
:SUMAR+;
:RESTAR—;
:MULTIPLICAR*;
```

para las próximas palabras aritméticas, y

```
: ESCRIBIR.;
```

para conseguir que nos aparezcan los resultados en la pantalla.

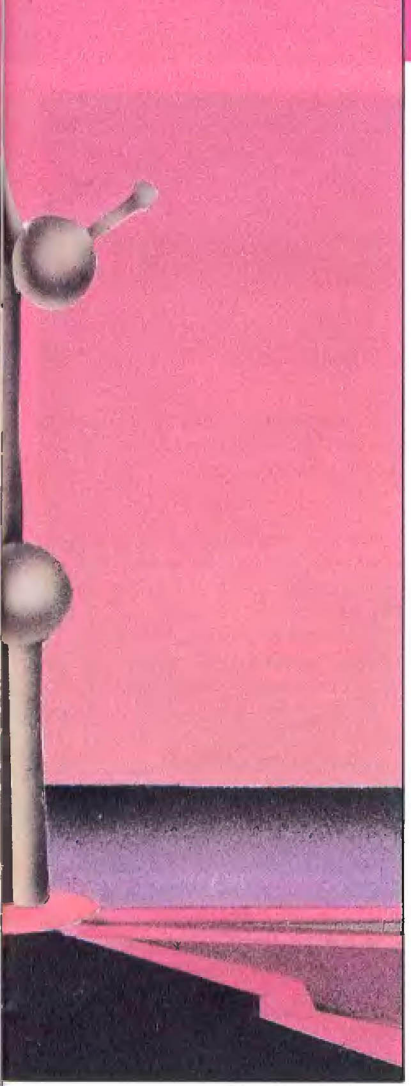
Con estas definiciones, la expresión aritmética que anteriormente analizamos podría transformarse en:

```
2 9 MULTIPLICAR 15 SUMAR ESCRIBIR
```

obteniendo el mismo resultado en ambos casos.

La primitiva expresión sigue siendo válida aunque actualmente hayamos definido otras, con nombre diferente, pero que realizan las mismas operaciones.

Todas las nuevas palabras Forth que que-



ramos crear se definen de la misma forma. Comenzamos poniendo dos puntos seguidos por el nombre que queremos darle, a continuación ponemos la serie de instrucciones que queremos agrupar y cerramos la definición con un punto y coma.

Los dos puntos indican al Forth que vamos a definir una nueva palabra y deben ir seguidos forzosamente por el nombre que queramos darle.

Después viene la secuencia de parámetros y palabras Forth ya definidas que se ejecutarán al llamar a la nueva. No es necesario que sean palabras standard del lenguaje —tales como los signos + y —del ejemplo anterior— sino que podemos usar también cualquier palabra de las que nosotros ya hemos creado.

Finalmente, la definición se cierra con un punto y coma. De esta manera podemos conseguir construir un lenguaje completo y compacto, en el que se emplean palabras standard del lenguaje para crear otras nuevas, y después utilizar las nuevas para otras sucesivas definiciones. El lenguaje crece de esta manera indefinidamente.

Los programas siguen el método de diseño «Top-down»

Así es exactamente como se genera un programa en Forth. Dividimos un problema general en grupos de acciones más particulares. Estos últimos ya se pueden definir utilizando las palabras standard Forth.

Y es en este punto donde comenzamos para luego ir definiendo otras cada vez más complejas. Al final llegaremos a un programa completo compuesto por una sola palabra que, al teclearla, hace que se ejecuten todas las asociadas.

Si seguimos usando el ejemplo anterior, podemos definir la siguiente palabra:

: CALCULO 2 9 MULTIPLICAR 15 SUMAR ESCRIBIR;

y ahora, con sólo teclear:

CALCULO

se ejecutarán todas las palabras agrupadas bajo este nombre, así como también las standard asociadas a ellas.

Por supuesto que los programas Forth no consisten únicamente en la evaluación de expresiones aritméticas —como en los casos anteriores— sino que también tienen a su disposición otras palabras que nos permitirán realizar cosas bastante más interesantes y atractivas.

Todas estas palabras básicas, incluidas en cada una de las versiones Forth que existen, es lo que se conoce como **«vocabulario»** del lenguaje, y en él están recogidas todas las que nos van a permitir utilizar variables, implementar bucles, explorar el teclado y un montón de cosas que son necesarias para que sus programas sean más complejos y le muestren la auténtica potencia del lenguaje.

Algunas órdenes del intérprete

Si quiere, puede hacer que le aparezca en la pantalla todo el vocabulario básico completo empleando el comando:

*VLIST

Y no sólo eso. También podemos visualizar todas las palabras que hayamos definido nosotros mismo. Si no lo ha hecho, le sugerimos que teclee las nuevas palabras de creación propia, para que por lo menos exista alguna.

A continuación teclee:

*LIST

y podrá conocer todas las que tiene a su disposición.

Pero empleando este comando puede también ver la serie de instrucciones que forman parte de cualquiera de ellas. Bastará con escribirlo seguido del nombre que queremos conocer y su listado estará **«servido»** .

¿Qué ocurre si por equivocación definimos dos veces una misma palabra? Sencillamente esta versión del Forth no acepta la segunda y nos da un mensaje de error:

Palabra ya definida

En otras, sí lo admite sin problemas. Guarda la segunda palabra además de la anterior. Pero cuando invocamos el nombre con el que está definida, el Forth accede a la más reciente de todas las que hayamos hecho sin tener en cuenta ninguna de las más antiguas.

Con este intérprete no se nos dará nunca este caso, ya que, como hemos dicho, no admitirá dos definiciones bajo el mismo nombre.

Entonces si queremos corregir o modificar una de las ya existentes el camino más cómodo a seguir sería listar la palabra para tener constancia de ella en la pantalla (no es necesario aunque sí muy conveniente). Después borrar del diccionario la que queramos cambiar mediante el comando:

*FORGET palabra

que le dirá al Forth que elimine de su diccionario la definición que hayamos hecho de «palabra».

A continuación creamos una nueva, auxiliándonos del listado existente en la pantalla con el cursor de copia, incluyendo las modificaciones que queramos hacer.

Sin embargo, para utilizar *FORGET debemos pensar muy bien lo que estamos haciendo ya que no sólo nos elimina la palabra que le hayamos dicho, sino que también borra todas las que estén definidas con posterioridad al nombre que sigue a *FORGET. Así que, ¡ojo con lo que borramos!

Hasta este momento debe tener por lo menos una pequeña idea de qué es lo que hay que hacer para trabajar con el Forth, pero seguro que todavía no tiene una noción clara de cómo podemos hacer un programa: no conocemos lo que hace cada una de las instrucciones del vocabulario standard del lenguaje.

Podemos dividirlas en grupos. El primero es el compuesto por todas las instrucciones encargadas de mantener en orden el stack cuando éste contiene parámetros. Son las instrucciones de tratamiento o manejo de la pila.

Instrucciones de manejo de la pila

Suponga que tenemos dos valores colocados en el stack y queremos hacer una división entre ellos. Por ejemplo:

6 2/.

nos calcularía el cociente entre 6 y 2 y nos lo sacaría en la pantalla.

Pero quizás el resultado que a nosotros nos interesaba es el inverso, o sea, el obtenido al

dividir 2 entre 6. La forma inmediata de hacerlo sería escribir:

2 6/.

No siempre es posible hacer esto, ya que en la mayoría de las ocasiones no conocemos los valores contenidos en el stack. Ahora bien, el Forth dispone de un operador que intercambia los dos valores superiores de la pila. En nuestro caso ya podríamos realizar el inverso de la operación anterior mediante:

6 2 SWAP/.

Para comprobar que con esta instrucción existe intercambio sin variar para nada el resto de los elementos del stack, teclee:

1 2 3 SWAP...

y aparecerán en la pantalla estos números pero con el 2 y el 3 cambiados de orden, o sea:

2 3 1

Pero no es este el único operador que maneja los elementos de la pila. ROT hace que giren los tres superiores. Coloca el tercero en la posición más alta y desplaza a los otros dos un lugar más abajo. Como siempre, la manera más efectiva de ver cómo funciona es un ejemplo práctico, así que escriba:

1 2 3 ROT...

El orden como quedan colocados es el que aparece en la pantalla y vemos que el 1 es el que está colocado encima de los otros dos. *¿Comprendido?*

Otro operador utilizado en Forth es DUP. Su misión es simplemente duplicar la última entrada que haya existido en el stack. Si quiere calcular el cuadrado de un número sería suficiente con hacer:

12 DUP *.

para que nos aparezca 144 (el cuadrado de 12) en la pantalla.

Si en lugar de esto lo que necesitamos es duplicar el elemento que está inmediatamente debajo del superior, el Forth posee un operador que realiza este trabajo: OVER. Comprúebelo tecleando:

1 2 OVER...

que copiaría el 1 encima de 2 (elemento superior en ese preciso instante) e imprimiría después los tres elementos existentes sacándolos de la pila y dejándola vacía.

El último de los operadores de manejo de stack es DROP, que saca del mismo la última entrada de datos. Ejemplo al canto:

1 2 DROP...

Y éstos son los más importantes. Su conocimiento y manejo pueden resultarnos un poco liosos al principio pero poco a poco nos iremos habituando a ello y podremos **«saborear»** sus ventajas.

Las estructuras del control del lenguaje son muy potentes

Pero el Forth no sólo dispone de operadores aritméticos o de manejo del stack, sino que además posee unas estructuras de control clásicas en otros lenguajes que le dan una gran potencia y versatilidad. Veámoslas.

La sentencia que nos permitirá alterar el orden de ejecución de comandos dependiendo de que se cumpla o no una determinada condición es IF... THEN... ELSE. Su forma general es:

< condición > IF < acción-1 >
ELSE < acción-2 >
THEN < continuar >

Su modo de funcionamiento es como sigue. La evaluación de la condición deja un «falso» o «verdadero» en el stack. Este resultado se analiza y si es verdadero se ejecuta la acción-1, si no, realiza la acción-2 y después continua el programa en el punto que sigue a THEN.

El Programa I sería un ejemplo de utilización de esta estructura. Para utilizarlo bastaría con teclear la nota seguida de la palabra **«NOTA»** una vez que haya sido definida para que se nos informe de si hay aprobado o no.

Programa I

```
: NOTA 5 < IF." SUSPENSO" ELSE." PROBADO" THEN CR." CORECTO ?";
```

El Forth dispone también de una serie de estructuras de bucle que hace que se repitan una serie de instrucciones durante un determinado número de veces. Es el conocido FOR... NEXT del lenguaje Basic.

< valor final >
< valor inicial > DO < acción > LOOP

Con él estamos ejecutando acción desde que el **«índice»** es igual al valor inicial hasta que alcanza el valor final.

La palabra clave LOOP hace que volvamos a DO mientras no se haya alcanzado el valor final. Le sugerimos que intente hacer variaciones sobre este pequeño ejemplo, Programa II, para clarificar sus ideas.

Programa II

```
: BUCLE1 12 0 DO. «ESTO ES UN BUCLE QUE SE REPITE 12 VECES» CR LOOP;
```

Podemos acceder al índice del bucle mediante la palabra Forth **«I»** que obtiene el valor en curso de la variable índice del bucle DO y le sitúa en la parte superior del stack. Si teclea el Programa III lo verá en funcionamiento.

Programa III

```
: BUCLE 13 1 DO I. SPACE. «ESTO ES UN BUCLE QUE SE REPITE 12 VECES» CR LOOP;
```

Si deseamos que el índice no se incremente de 1 en 1 sino con un valor diferente, se puede utilizar la estructura:

DO... número + LOOP

poniendo en **«número»** el incremento deseado, tanto positivo como negativo. Pero ésta es una facilidad que no contempla nuestro intérprete Forth. ¡Es una pena!

Existen también otra clase de bucles que van a repetir una serie de comandos dependiendo de que se cumpla o no una condición. En Forth este tipo de bucle puede ser de dos formas.

He aquí la primera de ellas:

El bucle WHILE nos va a permitir repetir una serie de acciones mientras se esté cumpliendo una determinada condición. Su forma general es:

BEGIN < condición > WHILE < acción >
REPEAT

La palabra BEGIN indica simplemente el punto donde comienza el bucle. Después la condición deja un valor en el stack dependiendo que se cumpla o no.

WHILE analiza dicho valor y si no es cero *(falso)* ejecuta la **«acción»**. REPEAT nos devuelve otra vez a BEGIN.

Si la condición no se cumple y, por tanto, el valor que coloca en el stack es cero, entonces no se ejecuta «acción» y se continúa por los siguiente a la palabra REPEAT.

El Programa IV es un ejemplo de la utilización de este tipo de estructuras.

Programa IV

```
: BUCLES 1 DUP BEGIN 13 < WHILE. «ESTO ES UN BUCLE QUE SE REPITE 12 VECES» CR 1 + DUP REPEAT;
```

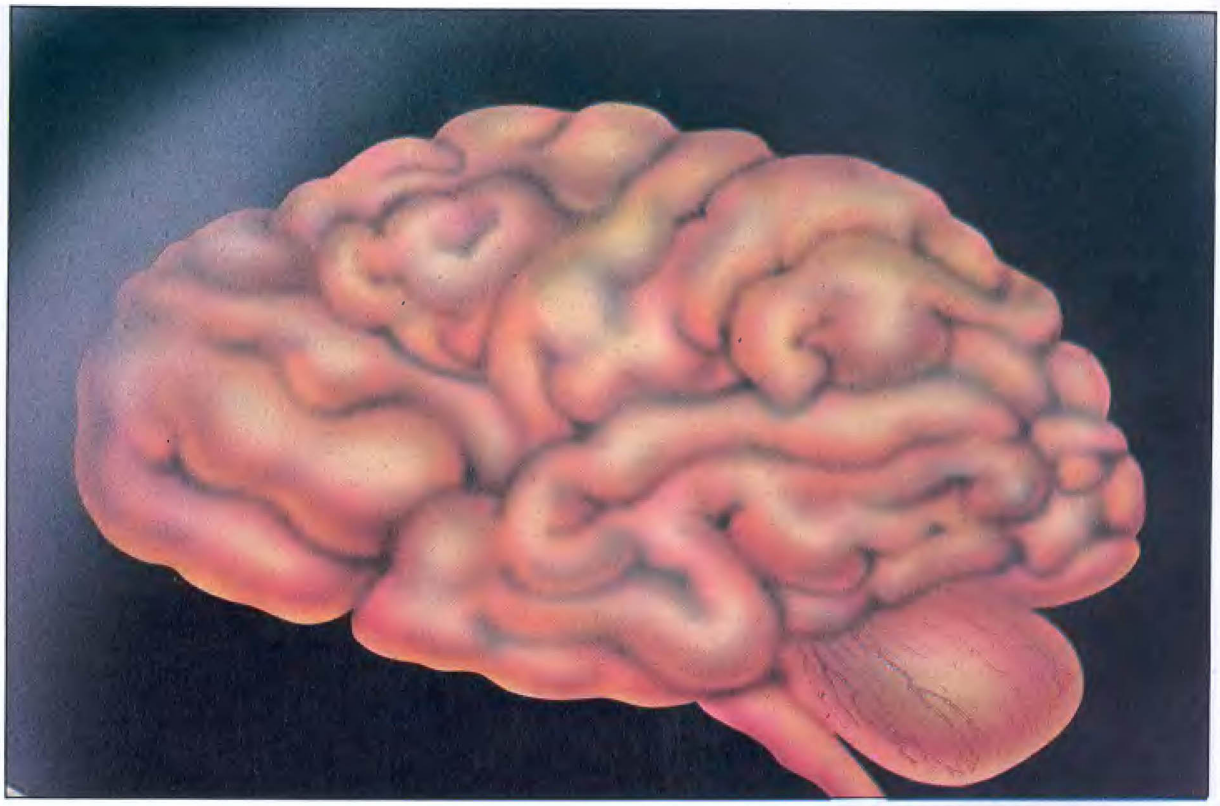
La segunda forma de bucle indefinido es la que hace que se esté ejecutando un proceso hasta que se cumpla una condición. Se trata de la instrucción GEGIN... UNTIL.

El formato más general de esta sentencia es:

BEGIN < acción > < condición > UNTIL

y su forma de trabajar es la siguiente:

BEGIN marca, como en el caso anterior, el comienzo del bucle. El cuerpo del mismo es



«acción» y será lo que se repita en el caso de no cumplirse la condición.

Después se evalúa la condición y deja en el stack un valor acorde con el resultado de la misma. UNTIL es la palabra clave que se encarga de encaminarnos a un sitio o a otro diferente. Cuando se encuentre con que el resultado lógico de la condición es falso *(cero)*, vuelve a BEGIN y repite el proceso.

Sin embargo, si es verdadero *(distinto de cero)* el programa continúa por lo que hay detrás de UNTIL.

Programa V

```
: BUCLE4 0 BEGIN. «ESTO ES UN BUCLE
QUE SE REPITE 12 VECES» CR 1+DUP
12=UNTIL. «FIN»;
```

Estos dos últimos bucles funcionan de una manera muy parecida: se evalúa una condición y dependiendo del resultado se continúa fuera del mismo o se repite la ejecución de un determinado proceso.

Pero existe una gran diferencia entre ambos: el momento donde se comprueba si se cumple la condición.

En el primer caso realizamos el análisis antes de ejecutar el cuerpo del bucle, por tanto, puede darse el caso que la condición sea falsa y no ejecutemos acción alguna.

Sin embargo, en el segundo siempre se va a ejecutar el proceso que hayamos definido entre BEGIN y UNTIL (al menos una vez). Se debe esto a que ahora la evaluación de la condición se hace al final del mismo. Así que, ¡cuidado con este matiz!

Parece que se nos ha olvidado algo al hablar de este lenguaje: las variables. ¿Es que no se emplean?

La verdad es que sí, existen y se usan. Pero una de las cosas que da potencia y velocidad al Forth es precisamente que utiliza las variables solamente cuando es imprescindible hacerlo.

Para pasar parámetros a rutinas o a operadores empleamos el stack tal y como lo hemos venido haciendo hasta ahora ya que se trata de valores temporales y no tiene mucho interés almacenarlos continuamente.

Las variables en Forth

Pero quizá necesitemos alguna vez guardar un determinado dato permanentemente. Y para esto están las variables en éste y en la mayoría de los lenguajes. Pero en este caso estarán definidas a nivel global, es decir, que pueden ser utilizadas por cualquiera de las palabras que estén en el vocabulario.

La forma de declararlas es:

< valor inicial > VARIABLE < nombre >

y el nombre que le hayamos dado se incorpora directamente al vocabulario de variables. Con:

```
0 VARIABLE PUNTOS
```

estamos creando una variable que se llama PUNTOS, a la que damos un valor inicial cero.

Podemos dar, o asignar, un valor distinto del inicial a una variable, ya definida, a lo largo de un programa. La forma de hacerlo en Basic sería:

```
PUNTOS=100
```

pero en Forth no se hace así. Es un poco diferente, compárelo usted mismo:

```
100 PUNTOS!
```

Y también podemos hacer el proceso inverso, es decir, recuperar en el stack el valor que previamente habíamos asignado a una variable.

Para colocar el valor de PUNTOS en el elemento superior del stack bastaría con hacer:

```
PUNTOS
```

y después imprimirlo con la palabra «.».

Pero estas dos últimas instrucciones se podrían agrupar en una sola. Si escribimos:

```
PUNTOS ?
```

obtendríamos el mismo resultado.

Por si queremos conocer todas las variables que hay definidas hemos implementado un comando que consigue hacerlas aparecer en la pantalla. Tecleando:

*VARLIST

podremos obtener todo el vocabulario de variables incorporadas al lenguaje Forth. Intente definir unas cuantas y compruebe cómo funciona esta nueva sentencia.

Ahora nos surge una pregunta: ¿qué hacemos cuando ya esté creado un programa completo compuesto por varias palabras, que funcione y cumpla su cometido?

El Forth es un lenguaje como todos los demás, luego lo que debemos hacer es guardar el programa en cinta o en disco para así poder volver a utilizarlo cuantas veces queramos.

Si quiere almacenarlo, o salvarlo, escriba:

*SAVE < nombre >

y todas las palabras y variables definidas que estén en ese momento en la memoria pasarán a un soporte físico (disco o cinta) y de esta forma no las perderemos.

Para volver a cargarlas otra vez en la memoria siga el mismo procedimiento al que ya está acostumbrado en Basic:

*LOAD < nombre >

Después de dar un repaso general a unas cuantas palabras standard del Forth, visualice el diccionario básico mediante:

*VLIST

Observará que en él hay incluidas algunas palabras que no son propias, precisamente, del lenguaje tales como CLG, DRAW, MOVE, etc.

Se han añadido a esta versión para darle una mayor potencia, sobre todo a la hora de crear gráficos y dibujos en la pantalla. Se utilizan de una forma muy similar a como se hace en Basic, pero teniendo en cuenta la nueva notación invertida que hemos aprendido.

Como muestra de sus posibilidades le dejamos el Programa VI. Echele un vistazo e intente explicarse cómo funciona.

Programa VI

```
: CUADRADO -2 -2 MOVER DUP 0 DRAWR 0 SWAP DRAWR DUP MINUS 0 DRAWR DUP MINUS 0 SWAPT DRAWR;
: FIGURA 0CLS 200 320 MOVE 1 DO I CUADRADO LOOP;
```

Y esto es todo por el momento. En una próxima oportunidad nos acercaremos un poco más a todas las palabras que utiliza este nuevo lenguaje Forth. Deseamos que, al menos, le llame un poquito la atención esta nueva forma de ver la programación.

```
10 REM ******************************
20 REM *       AMSTRAD FORTH        *
30 REM *                            *
40 REM *     MICROHOBBY AMSTRAD     *
50 REM ******************************
60 REM
70 REM INICIALIZACION
80 MODE 2:BORDER 13:INK 0,13:INK 1,
0
90 forth$=CHR$(12)+"Microhobby Fort
h V1.1"
100 OPENOUT "dummy":MEMORY HIMEM-1:
CLOSEOUT
110 temp=0:DEFINT a-z
120 DIM w$(130),p(130),beg(40),ff(4
0),df(40),bucle(40),ll(40),li(40)
130 er$(0)="OK":er$(1)="Desbordamie
nto de la capacidad minima del stac
k":er$(2)="Stack vacio"
140 er$(3)=" ya definido":er$(4)="
- nombre de variable ilegal":er$(5)
=" - Palabra no permitida":er$(6)="
Stack lleno":er$(7)="Stack de retor
no lleno"
150 spmax=100:DIM s(spmax):sp=-1
160 cvn=58:DIM cvoc$(cvn),dsp(cvn,1
)
170 FOR i=0 TO cvn:READ cvoc$(i),ds
p(i,0),dsp(i,1):NEXT
180 cvoc$(5)="."+CHR$(34)
190 umax=100:vmax=100:DIM uvoc$(uma
x),uvex$(umax),var$(vmax),ver(vmax)
200 uvn=-1:vrn=-1:PRINT forth$:PRIN
T:PRINT er$(0)
210 ON ERROR GOTO 2500
220 REM ENTRADA DE COMANDOS
230 w$="":er=0:LINE INPUT ln$:IF ln
$="" THEN 660 ELSE IF LEN(ln$)>240
THEN PRINT "Linea demasiado larga":
GOTO 230
240 WHILE ASC(ln$)=32:IF LEN(ln$)>1
 THEN ln$=RIGHT$(ln$,LEN(ln$)-1):WE
ND ELSE 660
250 WHILE RIGHT$(ln$,1)=CHR$(32):ln
$=LEFT$(ln$,LEN(ln$)-1):WEND
260 ln$=UPPER$(ln$):IF ASC(ln$)=ASC
("*") THEN IF LEN(ln$)>1 AND LEFT$(
ln$,2)<>"* " THEN GOSUB 1670:IF er
THEN 230 ELSE w$="":GOTO 660
270 ln$=ln$+CHR$(32):x$="":q=1:wn=-
1:comp=0
280 WHILE q<LEN(ln$)
290 p=q:WHILE MID$(ln$,q,1)<>" ":q=
q+1:WEND
300 w$=MID$(ln$,p,q-p):IF w$=":" TH
EN IF wn=-1 AND RIGHT$(ln$,2)="; "
THEN comp=-1:GOTO 610 ELSE PRINT "D
efinicion incorrecta":GOTO 230
310 FOR i=cvn TO 0 STEP-1:IF cvoc$(
i)<>w$ THEN NEXT:GOTO 370
320 IF comp THEN IF wn=0 THEN er=3:
GOTO 660
330 IF wn>=0 THEN IF w$(wn)="VARIAB
LE" THEN er=4:GOTO 660
340 x$=x$+CHR$(0)+CHR$(i+14):IF w$<
>"."+CHR$(34) THEN 610
350 te=INSTR(q,ln$,CHR$(34)+CHR$(32
)):IF te=0 THEN PRINT".";CHR$(34);"
 sin ";CHR$(34):GOTO 230
360 x$=x$+MID$(ln$,q+1,te-q-1)+CHR$
(4):q=te+1:GOTO 610
370 FOR i=uvn TO 0 STEP-1:IF uvoc$(
i)<>w$ THEN NEXT:GOTO 410
380 IF comp THEN IF wn=0 THEN er=3:
GOTO 660
390 IF wn>=0 THEN IF w$(wn)="VARIAB
LE" THEN er=4:GOTO 660
400 x$=x$+CHR$(i)+CHR$(i+14):GOTO 6
10
410 FOR i=vrn TO 0 STEP -1:IF var$(
i)<>w$ THEN NEXT:GOTO 450
420 IF comp THEN IF wn=0 THEN er=3:
GOTO 660
430 IF wn>=0 THEN IF w$(wn)="VARIAB
LE" THEN er=3:GOTO 660
440 x$=x$+CHR$(2)+CHR$(i+14):GOTO 6
10
450 FOR i=1 TO LEN(w$)
460 IF i=1 AND (ASC(w$)=ASC("+") OR
 ASC(w$)=ASC("-")) AND LEN(w$)>1 TH
EN 480
470 IF MID$(w$,i,1)<"0" OR MID$(w$,
i,1)>"9" THEN 530
480 NEXT i
490 IF comp AND wn=0 THEN er=5:GOTO
 660
500 IF wn>=0 THEN IF w$(wn)="VARIAB
LE" THEN er=5:GOTO 660
510 IF VAL(w$)>32767 OR VAL(w$)<-32
767 THEN PRINT"Numero ";w$;" demasi
ado grande":GOTO 230
520 x$=x$+CHR$(3)+w$+CHR$(4):GOTO 6
10
530 IF wn<0 THEN 560 ELSE IF w$(wn)
<>"VARIABLE" THEN 560
540 IF comp AND w$=w$(1) THEN er=3:
GOTO 660
550 x$=x$+w$+CHR$(4):GOTO 610
560 IF w$<>";" THEN 580
570 IF comp=0 OR q<>LEN(ln$) THEN P
RINT"Punto y coma ilegal":GOTO 230
ELSE 610
580 IF comp THEN IF wn=0 THEN 610
590 IF comp THEN IF w$<>w$(1) THEN
er=5:GOTO 660 ELSE x$=x$+CHR$(1)+CH
R$(uvn+15):GOTO 610
600 er=5:GOTO 660
610 wn=wn+1:w$(wn)=w$
620 WHILE MID$(ln$,q,1)=" ":q=q+1:W
END
630 WEND
640 x$=x$+CHR$(13):IF comp THEN 690
 ELSE 730
650 REM RUTINA DE ERROR
660 IF POS(#0)>1 THEN PRINT CHR$(32
);
670 PRINT w$;er$(er):GOTO 230
680 REM COMPILAR NUEVA PALABRA
690 IF wn<3 THEN PRINT"Definicion i
ncompleta":GOTO 230
700 uvn=uvn+1:uvoc$(uvn)=w$(1):uvex
$(uvn)=x$
710 w$="":GOTO 660
720 REM EJECUCION DE COMANDOS
730 ln=0:w$(ln)=x$:er=0
740 GOSUB 750:w$="":GOTO 660
750 p(ln)=1:ff(ln)=0:df(ln)=0
760 WHILE MID$(w$(ln),p(ln),1)<>CHR
$(13)
770 clase=ASC(MID$(w$(ln),p(ln),1))
:p(ln)=p(ln)+1
780 IF clase<>0 THEN 890
790 pal=ASC(MID$(w$(ln),p(ln),1))-1
4:p(ln)=p(ln)+1
800 IF ff(ln)=0 OR pal=37 OR pal=39
 OR pal=40 THEN 830
810 IF pal=5 OR pal=32 THEN WHILE A
SC(MID$(w$(ln),p(ln),1))<>4:p(ln)=p
(ln)+1:WEND:p(ln)=p(ln)+1
820 GOTO 1030
830 IF sp+dsp(pal,1)<-1 OR sp+dsp(p
al,1)>spmax THEN er=1:GOTO 1040
840 IF sp-dsp(pal,0)<-1 THEN er=2:G
OTO 1040
850 sp=sp+dsp(pal,1)
860 IF pal<43 THEN ON pal+1 GOSUB 1
060,1080,1090,1100,1110,1120,1130,1
140,1150,1160,1170,1180,1190,1200,1
210,1220,1230,1240,1250,1260,1270,1
280,1290,1300,1310,1320,1330,1340,1
350,1360,1370,1380,1390,1400,1410,1
420,1430,1440,1450,1460,1470,1480,1
490
870 IF pal>42 THEN ON pal-42 GOSUB
1500,1510,1520,1530,1540,1550,1560,
1570,1580,1590,1600,1610,1620,1630,
1640,1650
880 IF er=0 THEN 1030 ELSE 1040
890 IF clase<>1 THEN 940
900 pal=ASC(MID$(w$(ln),p(ln),1))-1
4:p(ln)=p(ln)+1
910 IF ff(ln) THEN 1030
920 IF ln<34 THEN ln=ln+1:w$(ln)=uv
ex$(pal) ELSE er=7:RETURN
930 GOSUB 750:IF ln=0 OR er=0 THEN
1030 ELSE RETURN
940 IF clase<>2 THEN 980
950 pal=ASC(MID$(w$(ln),p(ln),1))-1
4:p(ln)=p(ln)+1
960 IF ff(ln) THEN 1030
970 sp=sp+1:s(sp)=@var(pal):GOTO 10
30
980 IF clase<>3 THEN er=1:GOTO 1040
990 p=p(ln):WHILE ASC(MID$(w$(ln),p
,1))<>4:p=p+1:WEND
1000 v=VAL(MID$(w$(ln),p(ln),p-p(ln
)+1)):p(ln)=p+1
```

```
1010 IF ff(ln) THEN 1030
1020 sp=sp+1:s(sp)=v
1030 WEND
1040 ln=ln-1:RETURN
1050 REM LISTA DE COMANDOS
1060 temp!=s(sp+1):IF temp!<0 THEN
temp!=temp!+65536
1070 POKE s(sp+2),temp!-256*INT(tem
p!/256):POKE s(sp+2)+1,INT(temp!/25
6):RETURN
1080 s(sp)=s(sp)*s(sp+1):RETURN
1090 s(sp)=s(sp)+s(sp+1):RETURN
1100 s(sp)=s(sp)-s(sp+1):RETURN
1110 PRINT s(sp+1);CHR$(8);:RETURN
1120 WHILE ASC(MID$(w$(ln),p(ln),1)
)<>4 :PRINT MID$(w$(ln),p(ln),1);:p
(ln)=p(ln)+1:WEND:p(ln)=p(ln)+1:RET
URN
1130 s(sp)=INT(s(sp)/s(sp+1)):RETUR
N
1140 temp=s(sp):s(sp)=INT(s(sp-1)/s
(sp)):s(sp-1)=s(sp-1)-s(sp)*temp:RE
TURN
1150 s(sp)=(s(sp)<0):RETURN
1160 s(sp)=(s(sp)=0):RETURN
1170 s(sp)=(s(sp)<s(sp+1)):RETURN
1180 s(sp)=(s(sp)=s(sp+1)):RETURN
1190 s(sp)=(s(sp)>s(sp+1)):RETURN
1200 temp!=PEEK(s(sp+1))+256*PEEK(s
(sp+1)+1):IF temp!>32767 THEN temp!
=temp!-65536:PRINT temp!;CHR$(8);:R
ETURN ELSE PRINT temp!;CHR$(8);:RET
URN
1210 temp!=PEEK(s(sp))+256*PEEK(s(s
p)+1):IF temp!>32767 THEN s(sp)=tem
p!-65536:RETURN ELSE s(sp)=temp!:RE
TURN
1220 s(sp)=ABS(s(sp)):RETURN
1230 s(sp)=s(sp) AND s(sp+1):RETURN
1240 s(sp)=PEEK(s(sp)):RETURN
1250 PRINT:RETURN
1260 RETURN
1270 s(sp)=s(sp-1):RETURN
1280 PRINT CHR$(s(sp+1));:RETURN
1290 in$=INKEY$:IF in$="" THEN 1290
 ELSE s(sp)=ASC(in$):RETURN
1300 s(sp)=MAX(s(sp),s(sp+1)):RETUR
N
1310 s(sp)=MIN(s(sp),s(sp+1)):RETUR
N
1320 s(sp)=-s(sp):RETURN
1330 s(sp)=s(sp) MOD s(sp+1):RETURN
1340 s(sp)=s(sp) OR s(sp+1):RETURN
1350 s(sp)=s(sp-2):RETURN
1360 PRINT " ";:RETURN
1370 PRINT USING "&";SPACE$(s(sp+1)
-256*INT(s(sp+1)/256));:RETURN
1380 temp=s(sp):s(sp)=s(sp-1):s(sp-
1)=temp:RETURN
1390 vrn=vrn+1:var(vrn)=s(sp+1):WHI
LE ASC(MID$(w$(ln),p(ln),1))<>4:var
$(vrn)=var$(vrn)+MID$(w$(ln),p(ln),
1):p(ln)=p(ln)+1:WEND:p(ln)=p(ln)+1
:RETURN
1400 s(sp)=s(sp) XOR s(sp+1):RETURN
1410 beg(ln)=p(ln):RETURN
1420 IF s(sp+1)=0 THEN p(ln)=beg(ln
):RETURN ELSE RETURN
1430 IF s(sp+1)<>0 THEN RETURN ELSE
 ff(ln)=-1:RETURN
1440 IF ff(ln) THEN ff(ln)=0:RETURN
 ELSE p(ln)=beg(ln):RETURN
1450 IF s(sp+1)<>0 THEN RETURN ELSE
 ff(ln)=-1:RETURN
1460 ff(ln)=0:RETURN
1470 ff(ln)=-1-ff(ln):RETURN
1480 FOR i=0 TO uvn:uvoc$(i)="":uve
x$(i)="":NEXT:uvn=-1:PRINT forth$:P
RINT
1490 FOR i=0 TO vrn:var$(i)="":var(
i)=0:NEXT:vrn=-1:RETURN
1500 temp=s(sp-2):s(sp-2)=s(sp-1):s
(sp-1)=s(sp):s(sp)=temp:RETURN
1510 IF NOT df(ln) THEN df(ln)=-1:b
ucle(ln)=p(ln):li(ln)=s(sp+1):li(ln
)=s(sp+2):RETURN ELSE RETURN
1520 li(ln)=li(ln)+1:IF li(ln)<li(l
n) THEN p(ln)=bucle(ln):RETURN ELSE
 df(ln)=0:RETURN
1530 s(sp)=li(ln):RETURN
1540 CLG s(sp+1):RETURN
1550 DRAW s(sp+2),s(sp+1):RETURN
1560 DRAWR s(sp+2),s(sp+1):RETURN
1570 temp!=FRE(""):IF temp!>32767 T
HEN s(sp)=temp!-65536:RETURN ELSE s
(sp)=temp!:RETURN
1580 MOVE s(sp+2),s(sp+1):RETURN
1590 MOVER s(sp+2),s(sp+1):RETURN
1600 PLOT s(sp+2),s(sp+1):RETURN
1610 PLOTR s(sp+2),s(sp+1):RETURN
1620 s(sp)=RND*32768:RETURN
1630 s(sp)=TEST(s(sp+1),s(sp)):RETU
RN
1640 s(sp)=TESTR(s(sp+1),s(sp)):RET
URN
1650 x=XPOS:y=YPOS:PLOT 800,800,s(s
p+1):MOVE x,y:RETURN
1660 REM PROCESO DE EDICION DE COMA
NDOS
1670 er=0:w$=""
1680 IF ln$="$VLIST" THEN FOR i=cvn
 TO 0 STEP -1: PRINT cvoc$(i),:NEXT
:PRINT:RETURN
1690 IF ln$<>"$LIST" THEN 1750
1700 FOR i=uvn TO 0 STEP -1
1710 PRINT uvoc$(i),
1720 IF INKEY$="" THEN 1740
1730 WHILE INKEY$="":WEND
1740 NEXT:PRINT:RETURN
1750 IF LEFT$(ln$,6)<>"$LIST " THEN
 2010
1760 w$=RIGHT$(ln$,LEN(ln$)-6)
1770 WHILE ASC(w$)=32:w$=RIGHT$(w$,
LEN(w$)-1):WEND
1780 FOR i=uvn TO 0 STEP -1:IF w$<>
uvoc$(i) THEN NEXT:PRINT w$;" - Pal
abra desconocida":er=1:RETURN
1790 x$=uvex$(i)
1800 WHILE ASC(x$)<>13
1810 clase=ASC(x$):x$=RIGHT$(x$,LEN
(x$)-1)
1820 IF clase<>0 THEN 1890
1830 pal=ASC(x$)-14:x$=RIGHT$(x$,LE
N(x$)-1)
1840 PRINT cvoc$(pal);" ";
1850 IF pal<>5 AND pal<>32 THEN 199
0
1860 WHILE ASC(x$)<>4:PRINT LEFT$(x
$,1);:x$=RIGHT$(x$,LEN(x$)-1):WEND
1870 x$=RIGHT$(x$,LEN(x$)-1):IF pal
=5 AND clase<>3 THEN PRINT CHR$(34)
;
1880 PRINT " ";:GOTO 1990
1890 IF clase<>1 THEN 1930
1900 pal=ASC(x$)-14:x$=RIGHT$(x$,LE
N(x$)-1)
1910 PRINT uvoc$(pal);" ";
1920 GOTO 1990
1930 IF clase<>2 THEN 1970
1940 pal=ASC(x$)-14:x$=RIGHT$(x$,LE
N(x$)-1)
1950 PRINT var$(pal);" ";
1960 GOTO 1990
1970 IF clase<>3 THEN PRINT er$(1):
GOTO 2000
1980 GOTO 1860
1990 WEND
2000 PRINT:RETURN
2010 IF LEFT$(ln$,8)<>"$FORGET " TH
EN 2100
2020 w$=RIGHT$(ln$,LEN(ln$)-8)
2030 WHILE ASC(w$)=32:w$=RIGHT$(w$,
LEN(w$)-1):WEND
2040 FOR i=uvn TO 0 STEP -1:IF w$<>
uvoc$(i) THEN NEXT:GOTO 2070
2050 FOR j=i TO uvn:uvoc$(j)="":exe
c$(j)="":NEXT
2060 uvn=i-1:RETURN
2070 FOR i=vrn TO 0 STEP -1:IF w$<>
var$(i) THEN NEXT:PRINT w$;" - Pala
bra desconocida":er=-1:RETURN
2080 FOR j=i TO vrn-1:var$(j)=var$(
j+1):var(j)=var(j+1):NFXT
2090 vrn=vrn-1:RETURN
2100 IF ln$="$SAVE" THEN 2440
2110 IF LEFT$(ln$,6)<>"$SAVE " THEN
 2260
2120 w$=RIGHT$(ln$,LEN(ln$)-6)
2130 WHILE ASC(w$)=32:w$=RIGHT$(w$,
LEN(w$)-1):WEND
2140 dp=INSTR(w$,".")
2150 IF dp=0 THEN w1$=w$:w2$="4TH"
ELSE w1$=LEFT$(w$,dp-1):w2$=RIGHT$(
w$,LEN(w$)-dp)
2160 IF w1$="" OR LEN(w1$)>8 OR LEN
(w2$)>3 THEN 2440
2170 IF w2$="" THEN w2$="4TH"
2180 w$=w1$+"."+w2$
2190 OPENOUT w$
2200 PRINT #9,uvn:PRINT #9,vrn
2210 FOR i=0 TO uvn:PRINT #9,uvoc$(
i):PRINT #9,LEN(uvex$(i))
2220 FOR j=1 TO LEN(uvex$(i)):PRINT
 #9,ASC(MID$(uvex$(i),j,1));:NEXT j
2230 NEXT i
2240 FOR i=0 TO vrn:PRINT #9,var$(i
):PRINT #9,var(i):NEXT
2250 CLOSEOUT:RETURN
2260 IF ln$="$LOAD" THEN 2440
2270 IF LEFT$(ln$,6)<>"$LOAD " THEN
 2440
2280 w$=RIGHT$(ln$,LEN(ln$)-6)
2290 WHILE ASC(w$)=32:w$=RIGHT$(w$,
LEN(w$)-1):WEND
2300 dp=INSTR(w$,".")
2310 IF dp=0 THEN w1$=w$:w2$="4TH"
ELSE w1$=LEFT$(w$,dp-1):w2$=RIGHT$(
w$,LEN(w$)-dp)
2320 IF w1$="" OR LEN(w1$)>8 OR LEN
(w2$)>3 THEN 2440
2330 IF w2$="" THEN w2$="4TH"
2340 w$=w1$+"."+w2$
2350 OPENIN w$
2360 INPUT #9,uvn:INPUT #9,vrn
2370 ERASE uvoc$,uvex$,var$,var
2380 DIM uvoc$(umax),uvex$(umax),va
r$(vmax),var(vmax)
2390 FOR i=0 TO uvn:INPUT #9,uvoc$(
i):INPUT #9,lux
2400 FOR j=1 TO lux:INPUT #9,temp:u
vex$(i)=uvex$(i)+CHR$(temp):NEXT j
2410 NEXT i
2420 FOR i=0 TO vrn:INPUT #9,var$(i
):INPUT #9,var(i):NEXT
2430 CLOSEIN:RETURN
2440 IF ln$<>"$VARLIST" THEN PRINT
"Comando desconocido o incompleto":
er=-1:RETURN
2450 FOR i=vrn TO 0 STEP -1
2460 w$=var$(i)+CHR$(32)+STR$(var(i
))+SPACE$(2):PRINT w$,
2470 IF INKEY$="" THEN 2490
2480 WHILE INKEY$="":WEND
2490 NEXT:PRINT:RETURN
2500 er=6:IF sp>100 THEN sp=100:RES
UME NEXT ELSE w$="":RESUME 660
2510 REM DATOS DE PALABRAS RESERVAD
AS
2520 DATA "!",2,-2,"$",2,-1,"+",2,-
1,"-",2,-1,".",0,-1,"?",0,0,"/",2,-
1,"/MOD",2,0,"0<",1,0,"0=",1,0,"<",
2,-1
2530 DATA "=",2,-1,">",2,-1,"?",1,-
1,"@",1,0,"ABS",1,0,"AND",2,-1,"C@"
,1,0,"CR",0,0,"DROP",1,-1,"DUP",1,1
2540 DATA "EMIT",1,-1,"KEY",0,1,"MA
X",2,-1,"MIN",2,-1,"MINUS",1,0,"MOD
",2,-1,"OR",2,-1,"OVER",2,1
2550 DATA "SPACE",0,0,"SPACES",1,-1
,"SWAP",2,0,"VARIABLE",1,-1,"XOR",2
,-1,"BEGIN",0,0,"UNTIL",1,-1
2560 DATA "WHILE",1,-1,"REPEAT",0,0
,"IF",1,-1,"THEN",0,0,"ELSE",0,0,"F
ORTH",0,0,"CLEAR",0,0,"ROT",3,0,"DO
",2,-2,"LOOP",0,0,"I",0,+1
2570 REM DATOS PARA PALABRAS AMSTRA
D
2580 DATA "CLG",1,-1,"DRAW",2,-2,"D
RAWR",2,-2,"FRE",0,1,"MOVE",2,-2,"M
OVER",2,-2
2590 DATA "PLOT",2,-2,"PLOTR",2,-2,
"RND",0,1,"TEST",2,1,"TESTR",2,1,"G
RAPEN",1,-1
```