WHILE

WRITELN

END

LENGUAJE
DE PROGRAMACIÓN

# PASCAL

VA

AR

HIL

BEGIN

# AMSTRAD

## CPC·464

# HiSoft PASCAL 4T

## Soft 155

Publicado por AMSOFT, una división de

Amstrad Consumer Electronics plc
Brentwood House
169 Kings Road
Brentwood
Essex

First edition 1984

Editado por **INDESCOMP, S.A.**
Avda. Mediterráneo, 9 - 28007 MADRID (ESPAÑA)



Traduce, compone e imprime: **CONORG, S.A.**

**I.S.B.N.: 84-86176-17-4**

**Depósito Legal: M-43269-1984**

# SECCION 0
# PRELIMINARES

## 0.0 Introducción

El programa Hisoft Pascal para el Amstrad CPC464 (HP464) es una implementación rápida, fácil de usar y potente, del lenguaje Pascal tal y como está definido en el Pascal User Manual and Report (Jensen/Wirth Second Edition).

Hay algunas omisiones sobre esta especificación y son las siguientes:

La variable colectiva de tipo FILE no ha sido implementada, aunque sí pueden almacenarse en cinta colecciones de datos con estructura de LISTA.

La variable colectiva de tipo RECORD, usada para estructuras con formato de FICHA, no puede tener una parte VARIANTe.
Las PROCEDUREs y las FUNCTIONs no son válidas como **parámetros** actuales, o argumentos transferidos entre **procedimientos** y **funciones.**

En la práctica encontrarás que estas omisiones no restringen en absoluto el funcionamiento.

Se han incluido muchas funciones y procedimientos adicionales para reflejar el entorno cambiante en que se usan los **compiladores,** entre ellos están POKE, PEEK, TIN, TOUT y ADDR. Además se han añadido posibilidades extra para la versión Amstrad CPC464 con el propósito de aprovechar las potentes facultades disponibles en esta computadora -por ejemplo, para admitir el tratamiento de eventos y sucesos (AFTER, EVERY, etc.).

Además, se ha incluido un paquete de **rutinas** para Gráficos con Tortuga, en la Cara 2 de la cinta en cassette suministrada. Estas rutinas están documentadas en el Apéndice 5.

También se suministra una lista de rutinas auto-documentadas para interrelacionar Hisoft Pascal con el programa del sistema CPC464 pregrabado en ROM. Una lista de esas rutinas se da en el Apéndice 6.

El **compilador** ocupa aproximadamente 12K de almacenamiento, mientras el **explotador** para el momento de la ejecución, ocupa aproximadamente 5K y el **editor** 2K; quedando por lo tanto, cerca de 20K para tus programas y datos. El Hisoft Pascal con un programa de **vanguardia** almacenado en la memoria de escritura-lectura del sistema, asumiendo el control del CPC464, desplazando de él al programa de **vanguardia** que lo gestiona normalmente: el BASIC. No es posible, por tanto, combinar el BASIC y el Hisoft Pascal; y tampoco debiera haber ninguna necesidad de ello, dado que todo lo que puedes hacer en BASIC, también puedes hacerlo en Pascal.

Para abrirte el apetito y para que adquieras 'sensibilidad' ante este nuevo lenguaje, te presentaremos ahora un breve ejemplo de cómo redactar, **compilar** y ejecutar un programa en Pascal -recuerda que no hay ningún sustituto del estudio cuidadoso a través de todas las secciones de este manual, y te urgimos a que lo hagas antes de usar el programa Hisoft Pascal para **explotar** tus propios programas.

En particular, te recomendamos que leas esta sección preliminar, la sección relativa al Editor (Sección 4) y que te ejercites a través de los programas de ejemplo que te damos en el Apéndice 4.

### 0.0.1 Instrucciones para la Implantación en memoria

Primeramente, has de alojar el compilador en la memoria de tu máquina. Para ello, coloca la cinta en cassette suministrada dentro de la lecto-grabadora con el rótulo 'Hisoft Pascal' hacia arriba, y aprieba **PLAY** en el cassette. Ahora teclea **CTRL** y **[ENTER]** (retén pulsado **CTRL** y pulsa la pequeña tecla azul **[ENTER]** simultáneamente).

Un breve programa **cargador** en BASIC, será traído a la memoria y comenzará a auto-ejecutarse por sí mismo, produciendo un mensaje en el que se te pide:

```
RAM top ([ENTER] to default)?
```

que establezcas la 'cima' de la memoria de escritura-lectura (la dirección superior), y se te ofrece la posibilidad de pulsar **[ENTER]** para que el sistema adopte el valor **prescrito para omisiones.** En este momento debes teclear un número decimal seguido de **[ENTER]**, o bien simplemente pulsar **[ENTER]** por sí sola. Si especificas un número, será el que se tome como la más alta dirección de la memoria que NO va a ser usada por el Pascal; todas las direcciones por debajo de ella serán usadas y por tanto, la información que contienen quedará **corrupta.** Si simplemente presionas **[ENTER]** por sí sola, como respuesta a esta pregunta, se adoptará el valor **prescrito** de 45312, con lo que tendrás el máximo espacio para tus programas en Pascal. En la mayoría de los casos, usando solamente **[ENTER]** será suficiente; sólo necesitas especificar un número si deseas reservar algún espacio en la 'cima' o parte superior de la memoria para alojar programas en código máquina que desees interrelacionar con el Pascal.

Una vez que hayas especificado la 'cima' de la memoria de escritura-lectura, o adoptado el valor prescrito, entras a trabajar con el Editor interno suministrado que te mostrará una pantalla de ayuda.

### 0.0.2 Ejemplo de Programa

Supongamos que hemos optado por el valor prescrito (pulsando **[ENTER]**) para la 'cima' de la memoria, que el sistema ya está implantado en la memoria, estamos en comunicación con el Editor que nos saluda con su **divisa** '>' que es su símbolo característico. Teclea en ese momento **I**, concluyendo con **[ENTER]**.

Aparecerá el número 10 en el borde izquierdo de la pantalla: es un **número de línea** que automáticamente el editor genera para evitarte trabajo. El editor continuará generando números de línea al comienzo de cada una, hasta que **salgas** del modo I (que corresponde a **Inserción**). Tienes pues ante tí el número 10; teclea ahora el programa que te mostramos, concluyendo cada línea con la pulsación de la tecla **[ENTER]** y recuerda que los números de línea te los suministra automáticamente el editor.

```
1Ø  program hanoi;
2Ø  var n : integer;
3Ø  procedure movedisk(sce,dest : integer);
4Ø  begin
5Ø  write(sce:1,' to ',dest:1,'. ')
6Ø  end;
7Ø  procedure move(n,sce,aux,dest : integer);
8Ø  begin
9Ø  if n=1 then movedisk(sce,dest)
1ØØ else
11Ø begin
12Ø move(n-1,sce,dest,aux);
13Ø movedisk(sce,dest);
14Ø move(n-1,aux,sce,dest)
15Ø end;
16Ø end;
17Ø (*MainBlock*)
18Ø begin
19Ø write('Number of Discs? ');
2ØØ read(n); writeln;
21Ø move(n,1,2,3)
22Ø end.
230 [ESC]
```

Observa el uso de un **comentarios** encerrado mediante **(* *)** en la línea 170 y recuerda que los punto-y-coma **(;)** al final de las **sentencias** en Pascal son importantísimos.

Ahora deberás haber regresado al modo de comandos para el editor, con la divisa > de nuevo en pantalla. Ahora teclea la letra **C** (de **Compilar**) y concluye el comando pulsando **[ENTER]**. Eso hará que se compile tu programa obteniendo uno en código máquina y además se te proporcione un listado de la compilación. Al final de la compilación, debiera aparecer el mensaje 'Run?' que te pregunta si quieres ejecutarlo -y si no aparece y lo que sí se te presenta en pantalla es '*ERROR*', debes pulsar la letra **E** para poder regresar a tratar con el **Editor** y seguida como siempre de la tecla **[ENTER]**. Si obtienes un error, debes comprobar cuidadosamente el programa cotejándolo con el listado mostrado y volviendo a teclear cualquier línea errónea: puedes cambiar una línea tecleando su número de línea, seguido de un espacio en blanco y luego el **cuerpo** de la línea, y concluirla pulsando la tecla **[ENTER]**. Luego compila de nuevo el programa usando la **C**.

Si no hay ningún error, responde a la pregunta 'Run?' pulsando la **Y** (Yes=Si). El programa ahora se ejecutará, y la primera cosa que haga es preguntarte '¿Cuántos discos?'. El programa es la solución al ejercicio **recursivo** conocido como las Torres de Hanoi, en que dispones de tres pinchos en los que hay colocados discos de diferente tamaño en uno de ellos -el de origen-, según orden de tamaño; y el objetivo es pasar todos los discos a otro pincho -el de destino- uno a uno, sin colocar uno de mayor diámetro encima de otro de menor diámetro, y aprovechándote del tercer pincho -el auxiliar-. Es difícil de resolver, y puedes comprobarlo por ti mismo usando monedas.

Dale ahora al programa el número de discos con el que quieres 'jugar', y pulsa **[ENTER]**. Digamos que quieres comenzar con tres discos sólo, así que teclea:

3**[ENTER]**

El programa te producirá un listado de los movimientos que tienes que efectuar para pasar los discos desde el origen hasta el destino. El mensaje 'Run?' volverá a aparecer en la pantalla cuando el programa termine de trabajar. Teclea '**Y**' o '**y**' para que sea el programa el que trabaje de nuevo, o pulsa cualquier otra tecla para volver al editor..

Y eso es de lo que se trata! ...muy sencillo.

Ahora, lee por favor el resto de las Secciones de este manual cuidadosamente!

## 0.1 Ambito de este manual

Este manual no pretende enseñarte Pascal; para ello debes estudiar la Guía Tutorial al Hisoft Pascal o cualquiera de los excelentes libros mencionados en la Bibliografía, si eres un principiante en la programación con Pascal.

Este manual es un **documento de referencia,** que detalla los rasgos particulares del Hisoft Pascal.

La Sección 1 da la sintaxis y la **semántica** esperada por el compilador.

La Sección 2 detalla los diversos **identificadores** predefinidos que están disponibles dentro del Hisoft Pascal, desde CONSTantes hasta FUNCTIONes.

La Sección 3 contiene información sobre las diversas opciones disponibles en la compilación y también sobre el **formato** de los comentarios.

La Sección 4 muestra cómo usar el **Editor de Línea** que es una parte integrante del HP464.

Las secciones anteriores deben ser leídas cuidadosamente por todos los usuarios.

El Apéndice 1 detalla los **mensajes de error** generados tanto por el compilador como por el explotador en tiempo de ejecución.

El Apéndice 2 enumera los identificadores predefinidos y las **palabras reservadas.**

El Apéndice 3 da detalles de la **representación interna** de los datos dentro del Hisoft Pascal -útil para aquellos programadores que desean emplearse a fondo.

El Apéndice 4 da algunos **ejemplos** de programas en Pascal -estúdialos si tienes algún problema al escribir programas en Hisoft Pascal.

El Apéndice 5 contiene detalles del programa suministrado en la Cara 2 de la cinta maestra para hacer Gráficos con Tortuga.

El Apéndice 6 enumera algunos **procedimientos y funciones** en Pascal que te serán útiles al permitirte interrelacionarte fácilmente con el programa del sistema CPC464 grabado en la ROM.

## 0.2 Compilación y Explotación

Para detalles de cómo crear, enmendar, compilar y ejecutar un programa en Pascal en el HP464, usando el Editor de Línea integrado en el sistema, mira la Sección 4 de este manual.

Una vez que has recurrido al **compilador**, te generará un listado de la forma:

```
xxxx nnnn  texto de la línea fuente
```

siendo: xxxx la **dirección** donde comienza el código objeto generado para esa línea.
nnnn es el **número de línea** con los ceros delanteros suprimidos.

Si una línea contiene más de 80 caracteres, el compilador insertará caracteres de **avance de línea** automáticamente, de manera que la longitud de una línea de programa nunca sea mayor de 80 caracteres.

El listado puede ser dirigido hacia una impresora, si lo precisas, usando la opción de compilación $P (por impresora = **P**rinter, tal y como se ve en la Sección 3).

Puedes hacer una pausa en el listado en cualquier momento, pulsando **cualquier** tecla. Usa a continuación la tecla **[ESC]**ape para regresar al editor o cualquier otra tecla para reanudar el listado.

Si se detecta un error durante la compilación, se mostrará el mensaje '*ERROR*' seguido de una flecha hacia arriba ('↑'), que señala el símbolo que generó el error, y seguida de un **número de error** (véase Apéndice 1). El listado se detendrá; pulsa '**E**' para regresar al editor y revisar la línea que te muestra; pulsa '**P**' para regresar al editor y revisar la línea previa (si es que existe); o cualquier otra tecla para continuar con la compilación.

Si el programa termina incorrectamente (e.g. sin la sentencia 'END.') se mostrará el mensaje 'No more text' para indicar que **no hay más texto** y el control será devuelto al editor.

Si el compilador se queda **sin espacio para tablas** te mostrará el mensaje 'No Table Space' y también entregará el control al editor. Puedes especificar un tamaño diferente para la tabla, usando el comando para el editor de **alteración.**

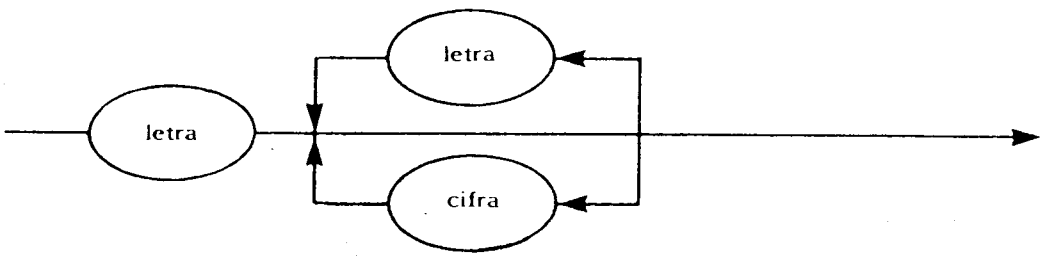
Si la compilación termina correctamente, pero contenía errores, el número de errores detectado será mostrado en pantalla y el **código objeto** no será generado por el compilador. Si la compilación transcurre con éxito, se mostrará el mensaje 'Run?' preguntándote si lo quieres 'ejecutar'; si deseas inmediatamente que el programa trabaje, responde con '**Y**' o '**y**' (Yes=Sí), y con otra letra cualquiera el control será devuelto al editor.

Durante la ejecución de un programa en **código objeto** pueden generarse diversos mensajes de error en **tiempo de ejecución** (véase Apéndice 1). Puedes suspender la **explotación** de un programa pulsando cualquier tecla; a continuación pulsa la tecla [**ESC**] para abandonar la operación, o cualquier otra tecla para reanudarla.

# SECCION 1
# SINTAXIS Y SEMANTICA

Esta sección detalla las normas sintácticas y el **significado** de las frases empleadas en el Hisoft Pascal -y a no ser que explícitamente se diga lo contrario, la implementación efectuada corresponde a la especificada en el Pascal User Manual and Report Second Edition (Jensen/Wirth).

## 1.1 IDENTIFICADOR

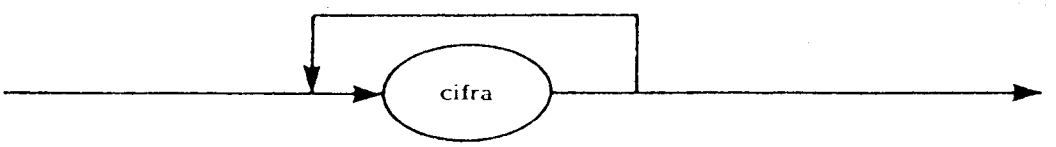



Sólo los primeros 10 caracteres de un identificador se tratan como **significativos.**

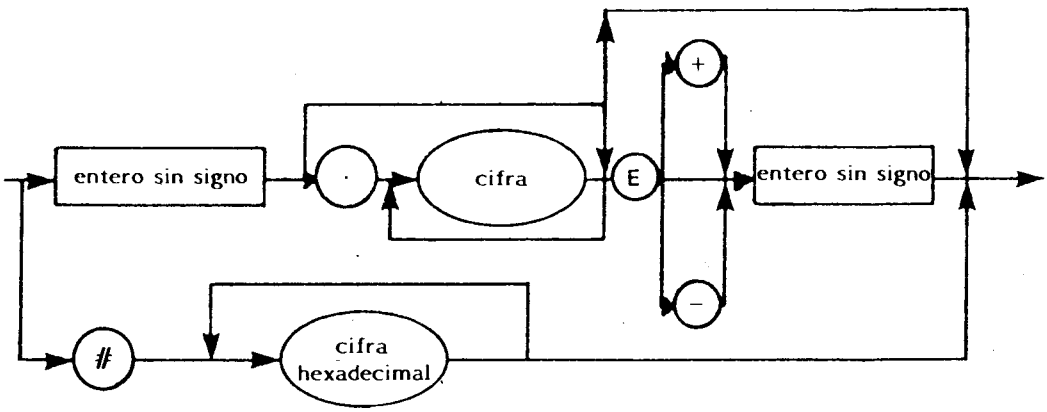
Los identificadores pueden contener letras en mayúsculas o en minúsculas. Las minúsculas serán convertidas a mayúsculas internamente, de manera que los identificadores ORIGEN, ORIGen y origen son equivalentes.

Las **palabras reservadas** y los identificadores predefinidos pueden imponerse por teclado en mayúsculas o en minúsculas, y son convertidos a mayúsculas por el editor.

## 1.2 Entero Sin-signo



## 1.3 Número Sin-signo




Los **enteros** tienen un valor absoluto menor o igual que 32767 en el Hisoft Pascal. Los números enteros mayores de ese valor absoluto se tratan como **reales.**

La **mantisa** de los reales tiene 23 bits de longitud. La precisión conseguida usando **reales** es por tanto de aproximadamente 7 cifras significativas. Observa que la precisión se pierde si el resultado de un cálculo es mucho menor que el valor absoluto de sus **argumentos,** e.g. 2.00002-2 no da como resultado de la resta el valor 0.00002. Eso es debido a la inexactitud involucrada al representar partes fraccionarias como **fracciones binarias.** Eso no ocurre cuando los enteros de magnitud moderada son representados como reales, e.g. 200002-200000=2 exactamente.

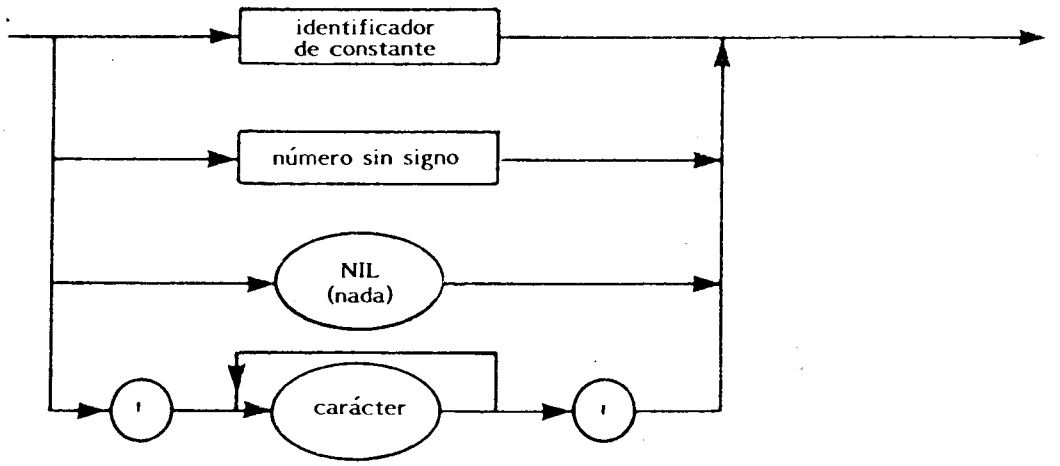
El **real** más grande disponible es 3.4E38 y el más pequéño es 5.9E-39.

No hay ninguna razón para usar más de 7 dígitos en la **mantisa** cuando se especifican **reales,** dado que los dígitos extra se ignoran excepto por el valor del lugar que ocupa.

Cuando la exactitud es importante, evita los ceros delanteros, dado que cuentan como uno de los dígitos. Por lo tanto 0.000123456 se representa con menos precisión de 1.23456E-4.

Los números en base 16 -**hexadecimales**- están disponibles para que los programadores especifiquen **direcciones de memoria** para interrelacionarse con programas **ensamblados,** entre otras cosas. Observa que debe haber como mínimo una cifra hexadecimal presente después del '#' porque en caso contrario se generará un error ('*ERROR* 51).

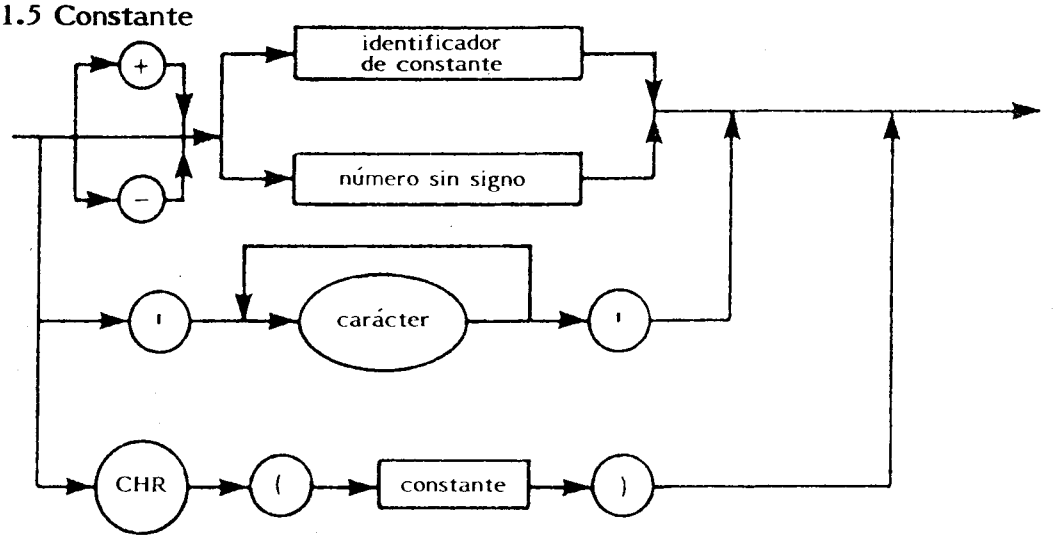
## 1.4 Constante Sin-signo




Observa que los literales (strings) no pueden contener más de 255 caracteres. Las estructuras del tipo **tabla** (array) de caracteres literales son `ARRAY [1..N] OF CHAR` siendo **N** un entero entre 1 y 255, ambos inclusive. Las constantes literales -las series o cadenas de caracteres consecutivos- no deben contener caracteres de **fin de línea** (`CHR(13)`) -si lo contienen, se generará un `'*ERROR*68'`.

Los caracteres disponibles son los valores ASCII del **repertorio** completo y ampliado hasta los 256 elementos. Para mantener la compatibilidad con el Pascal Standard, el carácter **nulo** no se representa como `" '' "`; en su lugar debe usarse `CHR(Ø)`.

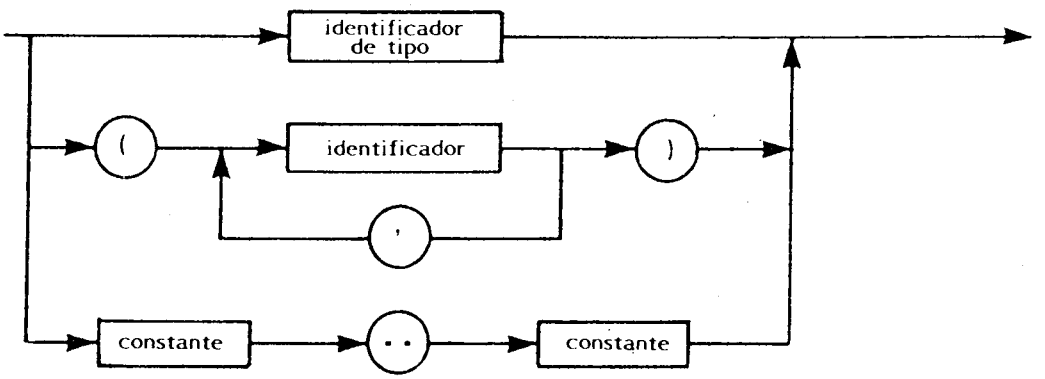
## 1.5 Constante




La **construcción** gramatical `CHR` que no es standard, se provee aquí para que puedan usarse constantes como caracteres de control. En este caso, la constante dentro del paréntesis debe ser del tipo **entero.**

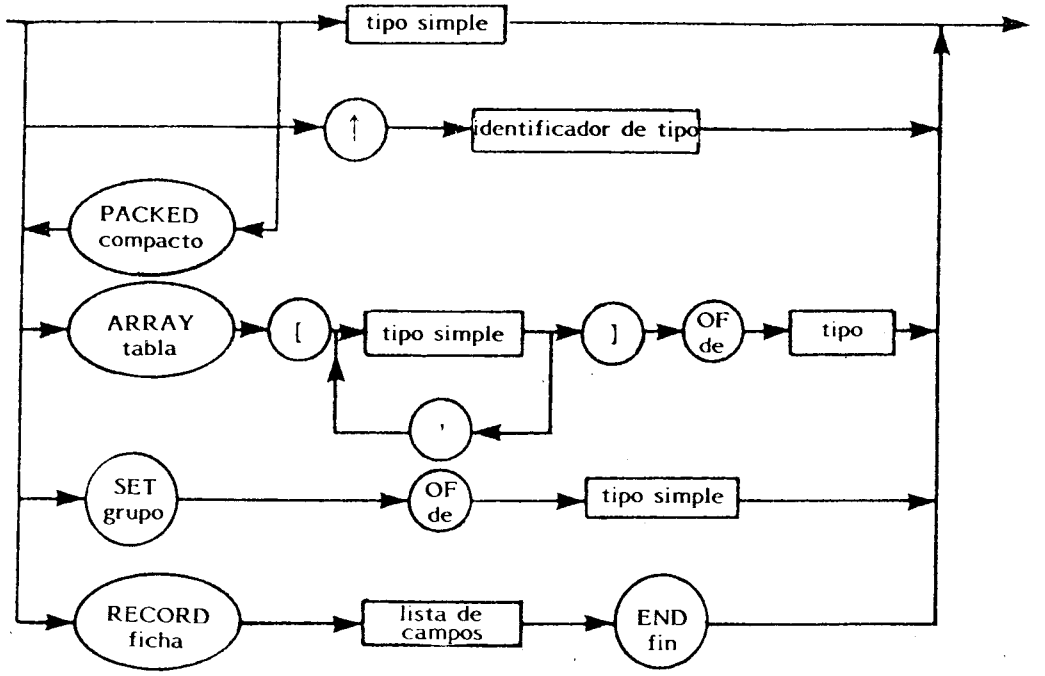
```
E.g. CONST   bs=CHR(8);      bs="backspace" (retroceso)
             cr=CHR(13);     cr="carriage return" (retorno de carro)
```

## 1.6 Tipo Simple




Los datos de tipo **scalar**, descritos por **enumeración** (identificador, identificador, ...) no pueden tener más de 256 valores distintos.

## 1.7 Tipo




La palabra reservada PACKED (=compacto) está aceptada pero se ignora dado que esa representación condensada interna siempre ocurre para representar los números y series de caracteres, etc. El único caso en que la notación compacta sería ventajosa, es con un Array (tabla) de Booleanos -pero eso se expresa de manera más natural como un Set (=grupo) cuando se requiere ahorrar espacio.

### 1.7.1 ARRAYs y SETs

El tipo básico de un SET (=grupo) puede tener hasta 256 elementos. Eso permite que se declaren SETs de CHAR (=grupos de caracteres) conjuntamente con grupos de **cualquier tipo** enumerados por el usuario. Observa sin embargo, que sólo pueden usarse como tipos base las sub-gamas de enteros. Todos los subgrupos de enteros se tratan como grupos de 0..255.

Se admiten 'arrays' completas de 'arrays', 'arrays' de grupos, fichas de grupos, etc.

Dos estructuras del tipo ARRAY sólo se tratan como equivalentes si su definición proviene del mismo uso de la palabra reservada ARRAY. Por lo tanto, los siguientes tipos no son equivalentes:

```
TIPO

tablaa=ARRAY[1..1ØØ] OF INTEGER;
tablab=ARRAY[1..1ØØ] OF INTEGER;
```

Por lo tanto, una variable del tipo 'tablaa' no puede asignarse a una variable del tipo 'tablab'. Eso permite detectar equivocaciones tales como asignar dos tablas que representan datos diferentes. La restricción anterior no se aplica al caso especial de 'arrays' de un tipo **serie,** dado que las 'arrays' de este tipo siempre se emplean para representar datos similares.

### 1.7.2 Punteros

El Hisoft Pascal permite la creación de **variables dinámicas** a través del uso del Procedimiento Standard NEW (véase Sección 2). Una variable dinámica, a diferencia de una variable estática, que tiene el espacio en memoria reservado para ella durante todo el bloque en que está declarado, no puede ser mencionada directamente mediante un identificador dado que no tiene tal identificador; en lugar de eso se emplea una **variable puntero.** Esta variable puntero, que sí es una variable estática, contiene la dirección de la variable dinámica y se puede conseguir acceso a la propia variable dinámica incluyendo una flecha ascendente (' ↑ ') después de la variable puntero. Pueden estudiarse ejemplos del uso de variables del tipo puntero en el Apéndice 4.

Hay algunas restricciónes en el uso de los **punteros** dentro del Hisoft Pascal. Son las siguientes:

No están permitidos los punteros hacia tipos que no hayan sido declarados. Eso no impide la construcción de estructuras de **lista** vinculadas, dado que las definiciones de tipo pueden contener punteros hasta ellos mismos, e.g.:

```
TYPE

item=RECORD
velue:INTEGER;
next: ↑ item
END;
link= ↑ item;      (link=eslabón, vínculo, engarce)
```

No se permite usar punteros que miren o señalen hacia punteros.

Los punteros hacia el mismo tipo se consideran como equivalentes, e.g.:

```
VAR
first:link;
current:↑ item;
```

Las variables 'first' y 'current' (primera y actual) son equivalentes (i.e. se usa la equivalencia estructural) y pueden asignarse una a otra, o compararse entre sí.

Se admite la constante predefinida NIL (=nada) y cuando se asigna a una variable puntero, se considera que dicha variable puntero no contiene **ninguna** dirección.

### 1.7.4 RECORDs

La implementación de **fichas** (que son variables estructuradas del tipo RECORD), consta de un determinado número de componentes que la constituyen y se denominan **campos,** dentro del Hisoft Pascal al igual que en el Pascal Standard, exceptuando que no se admite la parte VARIANTe de la lista de campos.

Dos datos del tipo RECORD sólo se tratan como **fichas** equivalentes si su declaración proviene del mismo uso de la palabra RECORD (véase Sección 1.7.1 anterior).

Observa que las declaraciones RECORD no abren un nuevo **ámbito** de aplicación de dicha declaración, y por tanto no debes usar el mismo identificador de campo en dos definiciones RECORD al mismo tiempo.

e.g. Si has declarado:

```
rec1 = RECORD
 f1:integer
END;
```

entonces no debes usar:

```
rec2 = RECORD
 f1:integer
END;
```

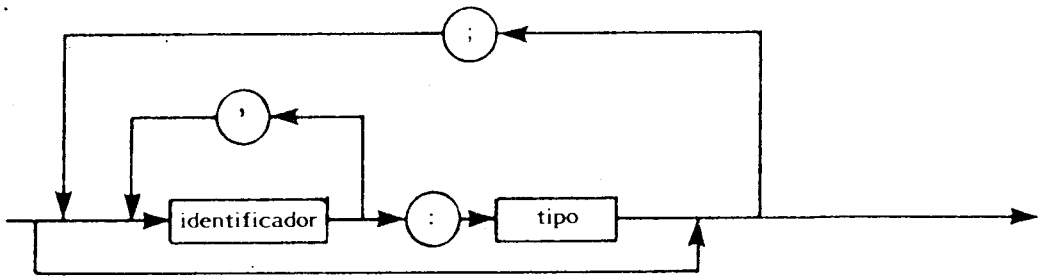
sino que debes usar:

```
rec2 = RECORD
 f2:integer
END;
```

La sentencia WITH (=con) puede usarse para conseguir el acceso a los diferentes campos de una ficha, en una forma más abreviada. Debes observar que la sentencia WITH no puede ser **citada recursivamente** y que WITH no abre un nuevo ámbito o esfera de utilización.

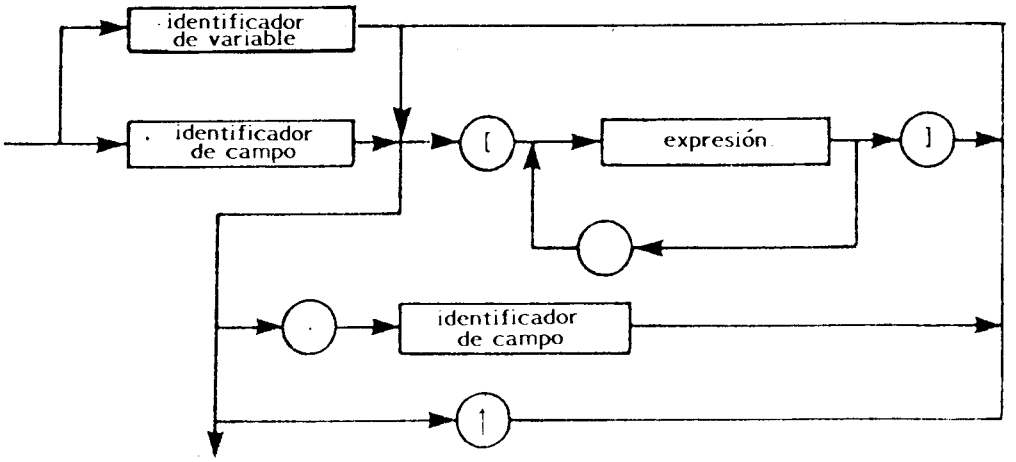
Véase el Apéndice 4 para un ejemplo del uso de WITH y RECORDs en general.

## 1.8 Lista de Campos




Usado para **fichas,** en conjunción con la declaración RECORD (véase la Sección 1.7.4 anterior y el Apéndice 4 para un ejemplo).

## 1.9 Variable




Se admiten dos categorías de variables en el Hisoft Pascal: variables **estáticas** y variables **dinámicas.** Las variables estáticas se declaran explícitamente mediante la palabra reservada VAR y se les reserva espacio en memoria durante toda la ejecución del bloque en que fueron declaradas.

Las variables **dinámicas,** por el contrario, se crean durante la ejecución del programa mediante el procedimiento NEW (=nueva). Estas variables no se declaran explícitamente y no se pueden mencionar, hacer **referencia** a ellas por un identificador. Se mencionan indirectamente a través de una variable estática del tipo **puntero,** que contiene la dirección de la variable dinámica.

Véase la Sección 1.7.2 y la Sección 2 para más detalles sobre el uso de las variables dinámicas, y el Apéndice 7 para un ejemplo.

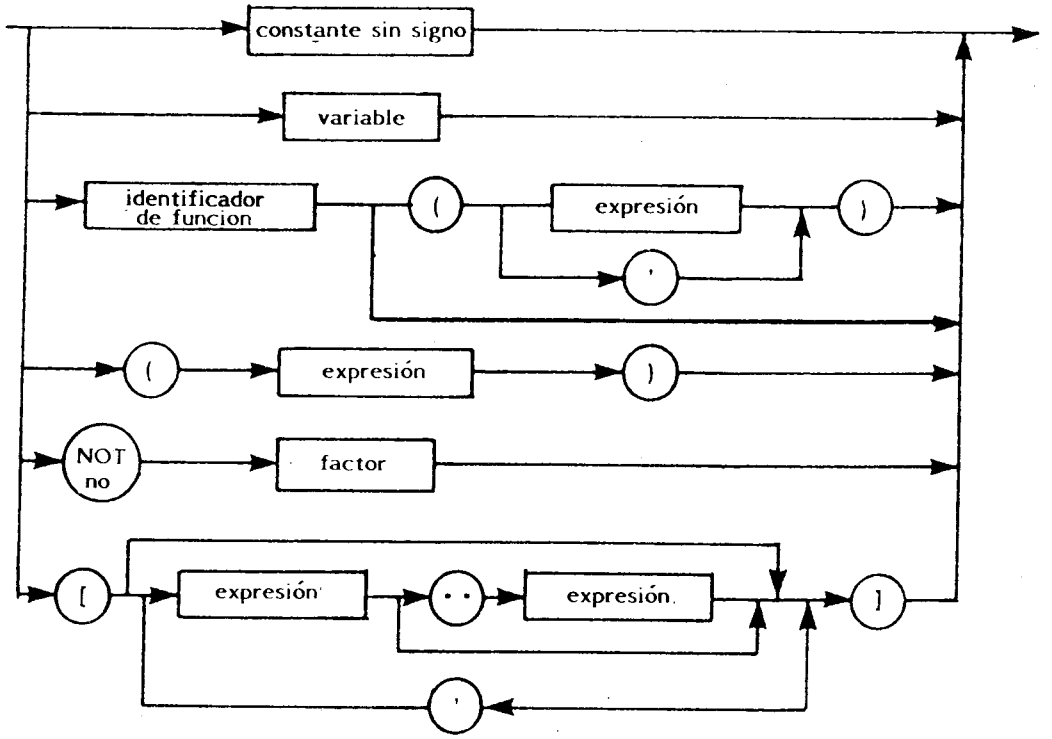
Al especificar elementos de una variable multi-dimensional, el programador no está forzado a usar la misma forma de especificación de **índice** al referirse a un elemento que la forma que usó en la declaración.

e.g. si la variable **a** se declara como

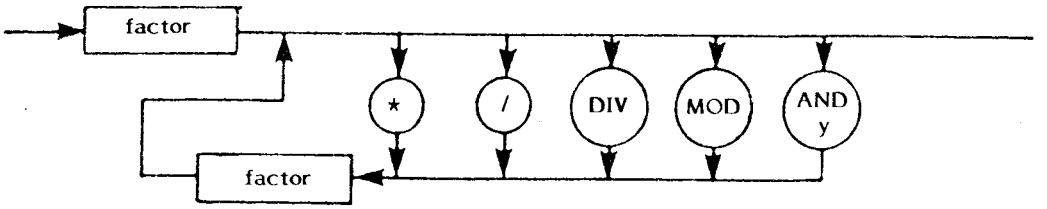
```
ARRAY[1..10] OF ARRAY[1..10] OF INTEGER
```

entonces se puede usar para referirse al elemento (1,1) de dicha 'array' entera, se puede usar la forma a `[1][1]` o bien la forma a `[1,1]`.

## 1.10 FACTOR

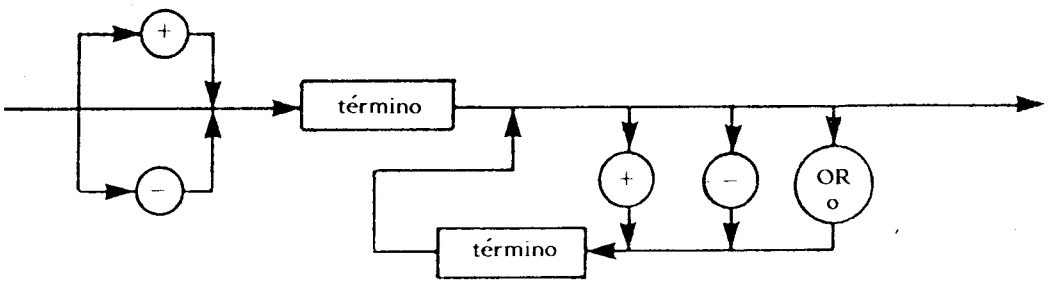



## 1.11 TERMINO



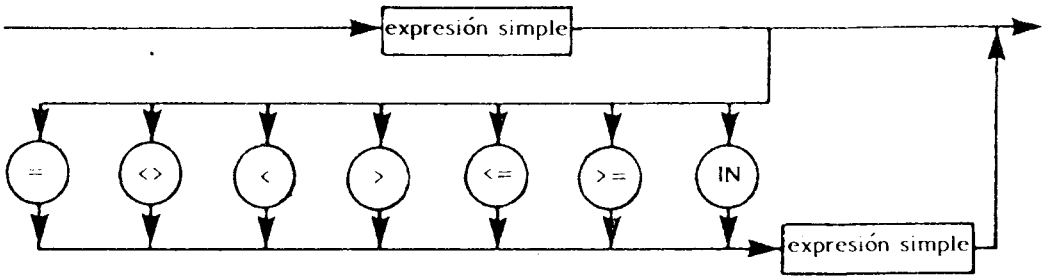

La cota inferior de un **grupo** (SET) es siempre cero y el tamaño del grupo es siempre el máximo del máximo valor del tipo base de dicho grupo. Así un `SET OF CHAR` siempre ocupa 32 bytes (dado que hay 256 elementos posibles y basta un bit por cada elemento). Similarmente un `SET OF Ø..1Ø` es equivalente a un `SET OF Ø..255`.

## 1.12 Expresión Simple




Los mismos comentarios efectuados en la Sección 1.11 concernientes a los grupos, se aplican también a las expresiones simples.

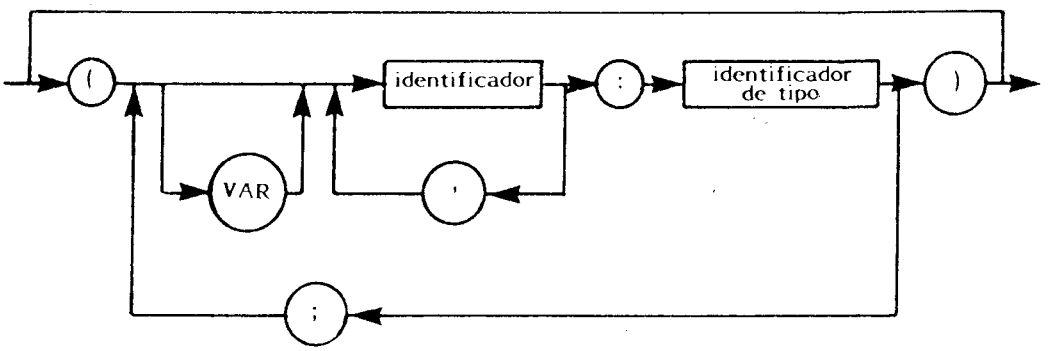
## 1.13 Expresión




Al usar el **operador de pertenencia** a un grupo `'IN'`, los elementos del grupo son todos los **subconjuntos** -incluyendo el conjunto total y el conjunto vacío- que puede formarse a patir de los elementos base del grupo; con la excepción de los grupos de enteros, para los que los elementos se consideran como si se hubiera definido `[0..255]`.

La sintaxis anterior se aplica al comparar **series de literales** de la misma longitud, **punteros** y todos los datos del tipo **scalar.** Los **grupos** pueden compararse usando >=, <=, < > o =. Los **punteros** sólo pueden compararse usando = y < >.

## 1.14 Lista de Parámetros




Un identificador de tipo debe ir seguido del signo dos-puntos, y en caso contrario se provocará el *ERROR* 44.

Se admiten completamente los parámetros **variables** así como los parámetros **constantes** (de valor).

Los procedimientos y las funciones no son válidos como parámetros (véase página 0.1).

## 1.15 Sentencias

Consulta el diagrama sintáctico de la siguiente página.

**Sentencias de asignación:**

Mira la Sección 1.7 para ver información sobre qué sentencias de asignación son ilegales.

**Sentencias** CASE:

No está permitido una lista de **selección** vacía al usar la sentencia "en caso de...". i.e. CASE OF END; generará un error (*ERROR* 13).

La cláusula ELSE, que es una alternativa para END en una sentencia CASE, se ejecuta en las demás situaciones. Es decir, cuando el valor obtenido en la expresión **selectora** no corresponde a ninguno de los valores previstos en la lista de constantes que marcan las acciones a realizar.

Si se usa END como terminador de la instrucción, y el valor del **selector** no corresponde con ninguno de la lista, el control se pasa a la sentencia que va inmediatamente detrás de la palabra clave END.

**Sentencias** FOR:

La variable de control de un bucle preconfinado establecido por la sentencia FOR sólo puede ser una variable elemental, no-estructurada; y tampoco un parámetro. Eso está a medio camino entre las definiciones standard de Jensen/Wirth y de ISO.

**Sentencias** GOTO:

Sólo es posible hacer un salto GOTO hasta un rótulo o etiqueta de sentencia que esté presente en el **mismo bloque** en que se encuentra la sentencia GOTO y al mismo nivel.

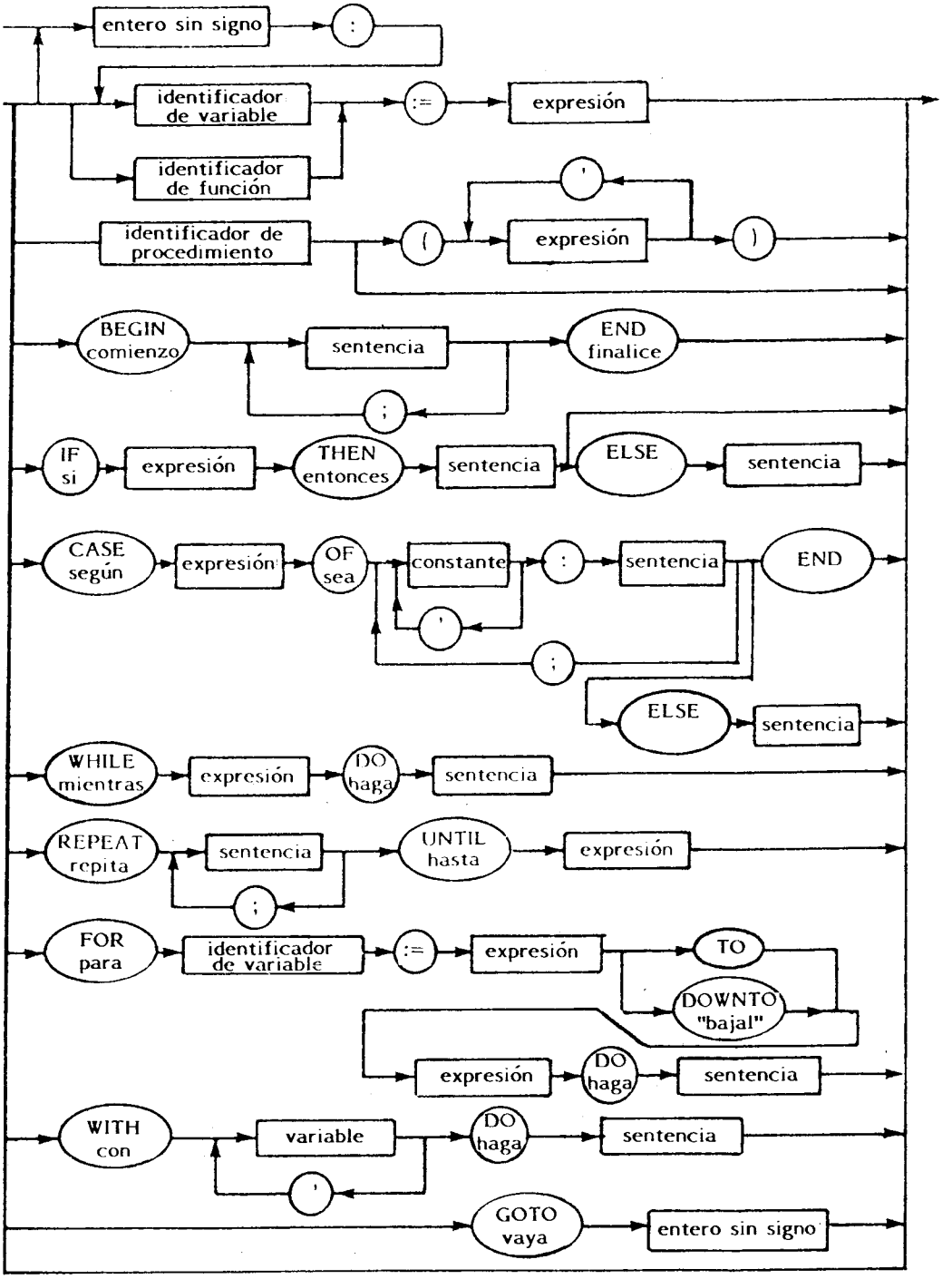
Los rótulos o etiquetas que designan una sentencia, deben declararse usando la palabra reservada LABEL, dentro del bloque en que se usan. Un rótulo consta de un mínimo de **uno** y un máximo de **cuatro** cifras. Cuando este **número de sentencia** se usa para designar una determinada sentencia, debe aparecer al comienzo de ella y debe ir seguido de un dos-puntos (':').

Observa que no debe usarse la sentencia GOTO para transferir la ejecución **fuera** de un bucle preconfinado FOR...DO... ni tampoco fuera de un procedimiento o de una función.

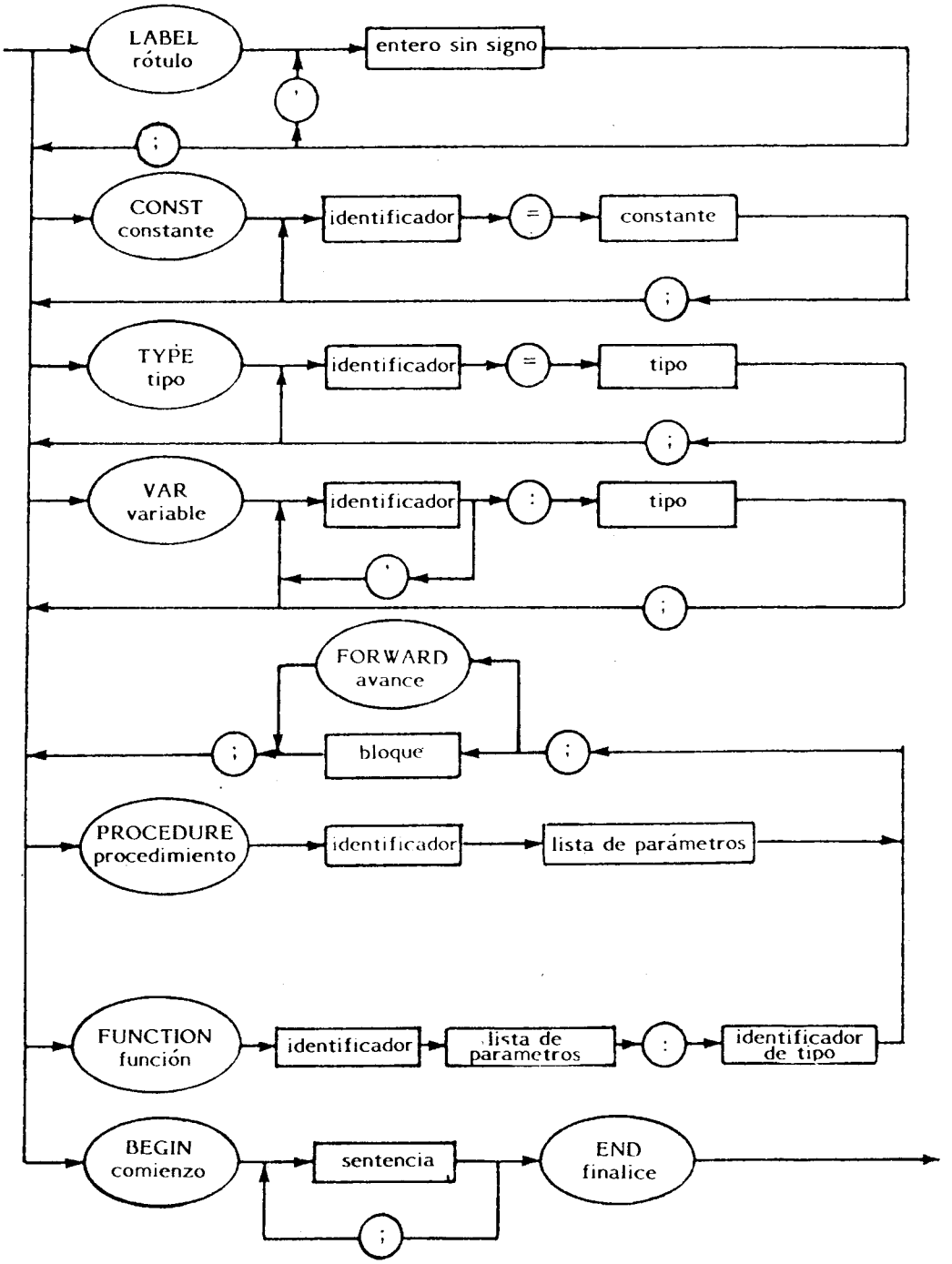
**Sentencias** WITH:

Las sentencias WITH no deben usarse **recursivamente** y no abren un nuevo ámbito de validez de la variable utilizada (véase Sección 1.7.4).

SENTENCIA:


entero sin signo
:
identificador de variable
:=
expresión
identificador de función
,
identificador de procedimiento
(
expresión
)
BEGIN comienzo
sentencia
END finalice
;
IF si
expresión
THEN entonces
sentencia
ELSE
sentencia
CASE según
expresión
OF sea
constante
:
sentencia
END
,
;
ELSE
sentencia
WHILE mientras
expresión
DO haga
sentencia
REPEAT repita
sentencia
UNTIL hasta
expresión
;
FOR para
identificador de variable
:=
expresión
TO
DOWNTO "baja!"
expresión
DO haga
sentencia
WITH con
variable
DO haga
sentencia
,
GOTO vaya
entero sin signo

## 1.16 Bloque


LABEL
rótulo
entero sin signo
,
;
CONST
constante
identificador
=
constante
;
TYPE
tipo
identificador
=
tipo
;
VAR
variable
identificador
:
tipo
,
;
FORWARD
avance
;
bloque
;
PROCEDURE
procedimiento
identificador
lista de parámetros
FUNCTION
función
identificador
lista de
parametros
:
identificador
de tipo
BEGIN
comienzo
sentencia
;
END
finalice

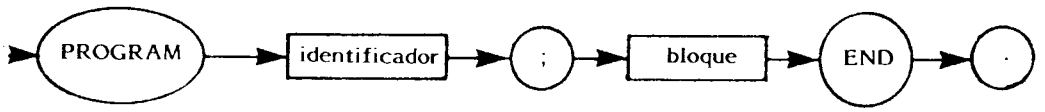
Referencias Anticipadas (Avance)

Al igual que en el Pascal User Manual and Report (Sección 11.C.1) pueden citarse los procedimientos y las funciones **antes** que hayan sido descritos si se declaran previamente usando la palabra reservada FORWARD, e.g.

```
PROCEDURE a(y:t) ; FORWARD;     (*avance de declaración del
PROCEDURE b(x:t);                procedimiento a*)
BEGIN
....
a(p);                           (*se cita el procedimiento a todavía
....                             no descrito*)
END;
PROCEDURE a;                    (*descripción real del procedimiento
BEGIN                            a*)
....
b(q);
....
END;
```

Observa que el tipo de parámetros y de resultado del procedimiento del que se hace un avance de declaración, mediante la palabra reservada FORWARD, no son repetidos cuando realmente se vuelve a declarar y se **describe** el procedimiento. Recuerda que FORWARD es una palabra reservada en Pascal.

## 1.17 Programa




Dado que la estructura FILE no se ha implementado, no hay parámetros formales en la declaración de PROGRAM, i.e. no necesitas (y desde luego, no debes) incluir (INPUT, OUTPUT) después del nombre del programa, tal y como se admite en el Pascal Standard.

## 1.18 TIPIficación Firme

Los diferentes lenguajes tienen diferentes maneras de asegurar que el usuario no emplea un elemento de información de manera que no sea coherente con la definición.

En un extremo de la escala de lenguajes está el **código máquina** donde no hay ninguna comprobación sobre el **tipo** de variable que se está mencionando. Detrás, tenemos un lenguaje como el Byte "Tiny Pascal" en que los caracteres, los enteros y los datos Booleanos se mezclan libremente sin generar errores. Avanzando más en la escala, viene el BASIC que distingue entre numerales y literales, y algunas veces entre enteros y reales (quizás usando el signo **%** para designar a los enteros).

Luego viene el Pascal que avanza mucho más al permitir tipos distintos **enumerados** por el usuario. En la 'cima' de la escala (por el momento) hay un lenguaje como ADA en que uno puede definir tipos numéricos diferentes, incompatibles.

Hay básicamente dos enfoques usados para las implementaciones de Pascal para reforzar la **tipificación**: equivalencia de estructura o equivalencia de nombres. El Hisoft Pascal usa equivalencia de nombres para fichas (RECORDs) y ARRAYs. La consecuencia de eso se clarifica en la Sección 1, pero bastará dar un ejemplo aquí: digamos que se definen dos variables múltiples en la forma siguiente:

```
VAR  A : ARRAY['A'..'C'] OF INTEGER;
     B : ARRAY['A'..'C'] OF INTEGER;
```

luego uno pudiera estar tentado de pensar que podría **hacer** A:=B; para asignar a A el valor de B, pero eso generaría un error (*ERROR* 1Ø) con el Hisoft Pascal, dado que se han creado mediante esas definiciones separadas dos estructuras de datos diferentes. En otras palabras, el usuario no ha adoptado la decisión de que A y B representaran el mismo **tipo** de información. Podría haberlo hecho mediante:

```
VARA,B : ARRAY['A'..'C'] OF INTEGER;
```

y ahora el usuario puede asignar libremente B a A y viceversa, dado que sólo se ha creado un **tipo** de dato.

Aunque superficialmente esta **equivalencia por nombre** puede parecer un poco complicada, en general lleva a menos errores en la programación, dado que exige más razonamiento inicial por parte del programador.

# SECCION 2
# IDENTIFICADORES PREDEFINIDOS

## 2.1 Constantes

| | |
|---|---|
| MAXINT | El entero más alto disponible, i.e. 32767 |
| FALSE, TRUE | Las constantes de tipo Booleano. |

## 2.2 Tipos

| | |
|---|---|
| INTEGER | Véase Sección 1.3. |
| REAL | Véase Sección 1.3. |
| CHAR | El repertorio de caracteres ASCII completo y ampliado a 256 elementos. |
| BOOLEAN | (FALSE, TRUE). Este tipo se usa en operaciónes lógicas incluyendo los resultados de comparaciones. |

## 2.3 Procedimientos y Funciones

### 2.3.1 Procedimientos de Entrada y Salida

2.3.1.1 WRITE (=escribir)

El procedimiento WRITE se usa para **sacar** información por la pantalla o por la impresora. Cuando la expresión que va a sacarse es simplemente del tipo carácter, entonces WRITE(e) **envía** el valor de 8 bits representado por el valor de la expresión **e** hasta la pantalla o hasta la impresora, según corresponda.

Observa:

| | |
|---|---|
| CHR(8) | **([CTRL]H)** da un retroceso **destructivo** en pantalla. |
| CHR(12) | **([CTRL]L)** limpia la pantalla o avanza página en la impresora. |
| CHR(13) | **([CTRL]M)** efectúa un retorno de carro con avance de línea. |
| CHR(16) | **([CTRL]P)** normalmente dirige el envío hacia la impresora si se usaba la pantalla, o viceversa. |

Sin embargo, generalmente:

```
WRITE(P1,P2,.......Pn);
```

es equivalente a:

```
BEGINWRITE(P1);WRITE(P2);........;WRITE(Pn)END;
```

Los datos a escribir P1,P2,....Pn pueden tener una de las siguientes formas:

```
<e> o <e:m> o <e:m:n> o <e:m:H>
```

siendo e,m y n expresiones, y H una constante literal.

Tenemos 5 casos que examinar:

1) **e** es del tipo entero: y se usa bien <e> o <e:m>.

El valor de la expresión entera **e** se convierte a una serie de caracteres con un espacio posterior. La longitud de la serie puede incrementarse (con espacios delanteros) por el uso de **m** que especifica el número total de caracteres a sacar. Si **m** no es suficiente para que **e** sea escrito, o **m** no está presente, entonces **e** se escribe por completo -con un espacio posterior- y **m** se ignora. Observa que si **m** se especifica como la longitud de **e** sin el espacio posterior, no se sacará ningún espacio detrás del dato numérico.

2) **e** es del tipo entero y se usa la forma <e:m:H>.

En este caso se saca **e** en hexadecimal. Si m=1 o m=2, el valor (e MOD 16 ↑ m) es el que se saca con una anchura de exactamente **m** caracteres. Si m=3 o m=4 se saca el valor completo de **e** en hexadecimal con una anchura de 4 caracteres. Si m>4 se insertan espacios delanteros antes del valor hexadecimal completo de **e** que se necesita. Los ceros delanteros se insertarán cuando corresponda. Ejemplos:

```
WRITE(1Ø25:m:H);

m=1  saca: 1
m=2  saca: Ø1
m=3  saca: Ø4Ø1
m=4  saca: Ø4Ø1
m=5  saca: _Ø4Ø1
```

3) **e** es del tipo real. Pueden usarse las formas <e>,<e:m> o <e:m:n>.

El valor de **e** se convierte a una serie de caracteres representando un número real. El formato de la representación está determinado por **n.**

Si **n** no está presente, se saca el número en notación científica, con una **mantisa** y un **exponente.** Si el número es negativo, se saca un signo menos precediendo a la mantisa, y en los demás casos se saca un blanco. El número siempre se saca con una posición decimal como mínimo y con un máximo de cinco posiciones decimales y el exponente siempre lleva signo (bien sea signo más o signo menos). Eso significa que la mínima anchura de la representación científica es de 8 caracteres; si la anchura del campo **m** es menor de 8, se sacará siempre la anchura completa de 12 caracteres. Si m>=8, se sacará una o más posiciones decimales hasta un máximo de 5 posiciones decimales (m=12). Para m>12, se insertan espacios delante del número. Ejemplos:

```
WRITE(-1.23E10:m);

m=7   saca:   -1.23ØØØE+1Ø
m=8   saca:   -1.2E+1Ø
m=9   saca:   -1.23E+1Ø
m=1Ø  saca:   -1.23ØE+1Ø
m=11  saca:   -1.23ØØE+1Ø
m=12  saca:   -1.23ØØØE+1Ø
m=13  saca:   _-1.23ØØØE+1Ø
```

Si se usa la forma <e:m:n> se escribirá la representación en **coma constante** del número **e,** con **n** especificando el número de posiciones decimales a sacar. No se sacará ningún espacio delantero a no ser que la anchura de campo **m** sea suficientemente grande. Si **n** es cero, se saca **e** como un entero. Si **e** es demasiado grande para ser sacado dentro de la anchura de campo especificada, se saca en notación científica con una anchura de campo de **m** (véase anteriormente). Ejemplos:

```
WRITE(1E2:6:2)        saca:     _ØØ.ØØ
WRITE(1E2:8:2)        saca:     __1ØØ.ØØ
WRITE(23.455:6:1)     saca:     __23.5
WRITE(23.455:4:2)     saca:     _2.3455ØE+Ø1
WRITE(23.455:4:Ø)     saca:     __23
```

4) **e** es del tipo carácter o tipo literal (string).

Se puede usar tanto <e> como <e:m> y el carácter o serie de caracteres se sacará en una anchura de campo mínima de 1 (pára caracteres) o la longitud de la serie (para tipos literales). Se insertan espacios delanteros si **m** es suficientemente grande.

5) **e** es del tipo Booleano.

Se puede usar tanto <e> como <e:m> y se sacará 'TRUE' (=cierto) o 'FALSE' (=falso) dependiendo del valor Booleano de **e,** usando un campo mínimo de 4 ó 5 posiciones respectivamente.

### 2.3.1.2 WRITELN (=escriba línea)

WRITELN da una nueva línea. Es equivalente a WRITE(CHR(13)). Observa que está incluido un **avance de línea.**

```
WRITELN(P1,P2,.....P3);
```

es equivalente a:

```
BEGINWRITE(P1,P2,.....P3);WRITELNEND;
```

### 2.3.1.3 PAGE (=página)

El procedimiento PAGE es equivalente a WRITE(CHR(12)); y hace que se limpie la pantalla o se avance la impresora hasta el principio de una nueva página.

### 2.3.1.4 READ (=leer)

El procedimiento READ se usa para **accesar** el teclado y recoger información. Lo hace a través de un depósito intermedio -un **'buffer' de entrada-** preparado durante el tiempo de ejecución. Este 'buffer' está inicialmente vacío (exceptuando una marca de fin de línea). Podemos considerar que cualquier acceso a este 'buffer' tiene lugar a través de una ventana de texto situada sobre el 'buffer', y a través de la cual podemos observar un carácter cada vez. Si esta ventana de texto está situada sobre una marca de fin de línea, entonces antes de que se termine la operación de lectura, se leerá una nueva línea de texto desde el teclado y se colocará dentro del 'buffer'.

Mientras está leyendo esta línea, todos los diversos códigos de control detallados en la Sección 0.0 serán reconocidos. Normalmente:

```
READ(V1,V2,.....Vn);
```

es equivalente a:

```
BEGINREAD(V1);READ(V2);.....;READ(Vn)END;
```

donde V1,V2, etc. pueden ser del tipo carácter, literal, entero o real.

La sentencia READ(V); tiene efectos diferentes dependiendo del tipo de dato que sea **V**. Hay 4 casos a considerar:

1) **V** es del tipo carácter.

En este caso READ(V) simplemente toma un carácter del 'buffer' de entrada y lo impone como valor de **V**.

Si la ventana de texto sobre el 'buffer' está situada sobre una marca de fin de línea (un carácter CHR(13)) la función EOLN -final de línea- entregará el valor TRUE (=cierto) y se leerá una nueva línea de texto desde el teclado sobre el 'buffer'. Cuando posteriormente se efectúe una operación de lectura, la ventana de texto estará situada sobre el comienzo de la nueva línea.

**Nota Importante:** Observa que EOLN es TRUE al comienzo del programa. Eso significa que si la primera lectura del teclado es del tipo carácter, se recogerá un valor CHR(13) y se efectuará a continuación la lectura de una nueva línea desde el teclado; una lectura posterior del tipo carácter sí entregará el primer carácter de esta nueva línea, suponiendo que no es un blanco. Véase también el procedimiento READLN explicado más adelante.

2) **V** es del tipo **serie** de caracteres.

Se puede leer del teclado una **serie** de caracteres usando READ, y en este caso se tomarán sucesivamente los caracteres de la serie, hasta que el número de caracteres definidos en ella haya sido tomado, o hasta que EOLN = TRUE. Si la serie de carácteres **V** no se rellena con los tomados por la operación de lectura (i.e. si se alcanza el final de línea antes de haber impuesto todos los caracteres posibles a la serie), el final de la variable literal se rellena con caracteres nulos (CHR(Ø)) -eso permite al programador evaluar la longitud de la serie que leyó.

La nota concerniente al párrafo 1) anterior, también se aplica aquí.

3) **V** es del tipo entero.

En este caso, la colección de caracteres que representa al entero, tal y como se definió en la Sección 1.3, es la leída del teclado. Se saltan todos los blancos precedentes y las marcas de fin de línea (lo que significa que los enteros pueden leerse inmediatamente -consúltese la nota en la sección 1) anterior).

Si el entero leído tiene un valor absoluto mayor de MAXINT (32767), se provocará el error en tiempo de ejecución 'Number too large' para indicar que **el número es demasiado grande** y se terminará la explotación del programa.

Si el primer carácter leído, después de haberse saltado los espacios y las marcas de fin de línea, no es una cifra o un signo (+ o -), se provocará el error en tiempo de ejecución 'Number expected' para indicar que se **esperaba un número** y se abandonará la explotación del programa.

4) **V** es del tipo real.

En este caso, se leerá la colección de caracteres sucesivos que representa a un número real de acuerdo con la sintáxis descrita en la Sección 1.3.

Todos los espacios delanteros y las marcas de fin de línea serán desechadas, y al igual que hemos mencionado para los enteros, el primer carácter que venga detrás debe ser una cifra o un signo. Si el número leído es demasiado grande o demasiado pequeño (véase Sección 1.3) se producirá un error de 'Overflow' para indicar que se ha **rebasado** o **desbordado** la capacidad permitida. Si se presenta una 'E' sin un signo o cifra detrás, se generará el error 'Exponent expected', y si hay presente un punto decimal sin una cifra a continuación, se dará el error 'Number expected'; para indicar respectivamente que era un **exponente** o un **número** el **esperado.**

Los reales, al igual que los enteros, pueden leerse inmediatamente. Véanse párrafos 1) y 3) anteriores.

### 2.3.1.5 READLN

```
READLN(V1,V2,.....Vn);
```

es equivalente a:

```
BEGIN READ(V1,V2,.....Vn); READLN END;
```

READLN simplemente recoge información del teclado y la mete en un nuevo 'buffer' de entrada; y mientras se introduce en el 'buffer' puedes usar las diversas funciones de control detalladas en la Sección 0.0. Por tanto, EOLN (final de línea) se hace FALSE (=falso) después de la ejecución de READLN a no ser que la siguiente línea esté en blanco.

El procedimiento READLN puede usarse para saltarse la línea en blanco que está presente al comienzo de la ejecución de un programa en código objeto, i.e. tiene el efecto de preparar un nuevo 'buffer' de entrada. Esto será útil si deseas leer un componente del tipo carácter al comienzo de un programa, pero no es necesario si estás leyendo uno entero o uno real (dado que se saltan las marcas de fin de línea) o si estás leyendo caracteres procedentes de las líneas posteriores.

En general, es mejor usar un simple READLN después de **avisar** para que se imponga el dato, y después citar el procedimiento READ.

Ejemplo de lectura de variables del tipo carácter:

```
PROGRAM READCHAR;
VAR CH:CHAR;
BEGIN
 REPEAT
  WRITE ('TECLEA ALGUNOS CARACTERES');
  READLN;
  WHILE NOT EOLN DO
   BEGIN
    READ (CH);
    WRITE ('El ASCII correspondiente a 'CH,' es
    ',ORD(CH))
    END;
 UNTIL CH='E'
END.
```

### 2.3.2 Funciones de Entrada

#### 2.3.2.1 EOLN (=fin de línea)

La función EOLN es una función Booleana que entrega el valor TRUE (=cierto) si el siguiente carácter recogido del teclado es un carácter de fin de línea CHR(13). En los demás casos, la función entrega el valor FALSE (=falso).

#### 2.3.2.2 INCH (=¿ingresado carácter?)

La función INCH hace que se examine el teclado de la computadora y -si ha sido pulsada una tecla- entregar el carácter correspondiente a la tecla pulsada. Si no ha sido pulsada ninguna tecla, entrega como resultado CHR(Ø). La función por tanto, da como resultado un dato de tipo carácter. Esta función deberá usarse con la opción **C** del compilador (véase Sección 3).

### 2.3.3 Funciones de Transferencia

#### 2.3.3.1 TRUNC(X)

El argumento **X** debe ser del tipo real o entero, y el valor entregado por esta función de **truncamiento** es el mayor de los enteros que es menor o igual a **X**, cuando **X** es positivo; o el ménor de los enteros que es mayor o igual que **X**, cuando **X** es negativo. Ejemplos:

TRUNC(-1.5) da -1, TRUNC(1.9) da 1

#### 2.3.3.2 ROUND(X)

El argumento **X** debe ser del tipo real o del entero, y esta función de **redondeo** da como resultado el entero **más próximo** a **X** (de acuerdo con las reglas de redondeo standard). Ejemplos:

ROUND(-6.5) da -6 ROUND(11.7) da 12
ROUND(-6.51) da -7 ROUND(23.5) da 24

#### 2.3.3.3 ENTIER(X)

El argumento **X** debe ser del tipo real o del entero, y la función de **entero máximo** entrega el entero mayor que es más pequeño que igual a **X**, para todos los valores de **X** positivos o negativos. Ejemplos:

```
ENTIER(-6.5)  da  -7        ENTIER(11.7)  da  11
```

**Nota:** La función ENTIER no es una función standard en Pascal, pero es el equivalente de la función INT del BASIC Amstrad. Es útil para escribir rutinas rápidas en muchas aplicaciones matemáticas.

#### 2.3.3.4 ORD(X)

El argumento **X** puede ser de cualquier tipo scalar, exceptuando el real. El valor entregado es un entero que representa el número **ordinal** que el valor **X** tiene dentro del conjunto de valores que puede adoptar la variable **X**.

Si **X** es del tipo entero, entonces ORD(X)=X; y por tanto, debe evitarse.

Ejemplos:

```
ORD('a')   da   97        ORD('@')   da   64
```

#### 2.3.3.5 CHR(X)

El argumento **X** debe ser del tipo entero. El valor entregado es el **carácter** que corresponde al valor de **X**, según el ASCII. Ejemplos:

```
CHR(49)   da   '1'        CHR(91)   da   '['
```

### 2.3.4 Funciones Aritméticas

En todas las funciones mencionadas en esta subsección, el argumento **X** de la función debe ser del tipo real o del tipo entero.

#### 2.3.4.1 ABS(X)

Entrega el valor **absoluto**, la magnitud de **X** (e.g. ABS(-4.5) da 4.5). El resultado es del mismo tipo que **X**.

#### 2.3.4.2 SQR(X)

Entrega el valor X*X, i.e. el **cuadrado** de **X**. El resultado es del mismo tipo que **X**.

#### 2.3.4.3 SQRT(X)

Entrega la **raíz cuadrada** de **X** -y siempre el resultado es del tipo real. Si el argumento **X** es negativo se genera un 'Maths CallError' para indicar que es un **error de cita** a las rutinas **matemáticas.**

#### 2.3.4.4 FRAC(X)

Entrega la parte **fraccionaria** de **X**, y por tanto FRAC(X)=X-ENTIER(X).

Al igual que con la función de **entero máximo,** esta función es útil para escribir rutinas matemáticas muy rápidas. Ejemplos:

FRAC(1.5) da Ø.5 FRAC(-12.56) da Ø.44

2.3.4.5 SIN(X)

Entrega el **seno** de **X,** estando **X** en radianes. El resultado es siempre del tipo real.

2.3.4.6 COS(X)

Entrega el **coseno** de **X,** estando **X** en radianes. El resultado es siempre del tipo real.

2.3.4.7 TAN(X)

Entrega la **tangente** de **X,** estando **X** en radianes. El resultado es siempre del tipo real.

2.3.4.8 ARCTAN(X)

Entrega el **arco** cuya **tangente** es el número **X.** El resultado es siempre del tipo real y corresponde a radianes.

2.3.4.9 EXP(X)

Entrega el **antilogaritmo** o **exponencial** de **X** (es decir e ↑ X siendo e=2.71828). El resultado es siempre del tipo real.

2.3.4.10 LN(X)

Entrega el **logaritmo** natural de **X** (tomando como base e=2.71828). El resultado es de tipo real. Si X<=Ø, generará un 'Maths Call Error' para indicar que es un **error de cita** a las rutinas **matemáticas.**

### 2.3.5 Procedimientos Predefinidos Avanzados

2.3.5.1 NEW(p)

El procedimiento NEW(p) adjudica espacio a una **nueva** variable dinámica, señalada por la variable p. Dicha variable p es una variable del tipo **puntero,** y después de haber sido ejecutado el procedimiento NEW(p), contiene la dirección donde está alojada la variable dinámica recién creada. El tipo de la variable dinámica es el mismo que el de la variable puntero p, y esta puede ser de cualquier tipo.

Para conseguir el **acceso a** la variable dinámica, se emplea la notación p ↑ como **referencia,** y se dice -leyendo de derecha a izquierda- que es la señalada o la **aludida** por p. (Véase el Apéndice 4 para un ejemplo del uso de **punteros** para referirse a variables dinámicas.

Para cambiar la adjudicación del espacio ocupado por las variables dinámicas, usa el procedimiento MARK para **marcar** el espacio acotado, y el procedimiento RELEASE para dejar libre dicho espacio.

### 2.3.5.2 MARCK(v1)

Este procedimiento conserva la **cota** alcanzada en el cúmulo de variables dinámicas apiladas hasta ese momento, guardándolo como valor de la variable **v1** de tipo puntero. El estado de dicho cúmulo, puede restaurarse dejándolo en la **cota** que tenía cuando se ejecutó el procedimiento MARK, mediante el procedimiento RELEASE (véase más adelante).

El tipo de la variable a la que señala **v1** no tiene importancia, dado que **v1** solamente se usará con los procedimientos MARK y RELEASE, pero nunca con NEW.

Para un ejemplo de programa que usa MARK y RELEASE, véase el Apéndice 4.

### 2.3.5.3 RELEASE(v1)

Este procedimiento **libera** espacio del cúmulo donde se apilan los valores de las variables dinámicas. El estado de dicho cúmulo se restaura a la situación que tenía cuando se ejecutó MARK(v1), que **marcó** en **v1** la **cota** que alcanzaba. Por lo tanto, este procedimiento "destruye" efectivamente (**-quita-**) todas las variables dinámicas creadas desde la ejecución más reciente del procedimiento MARK. Por lo tanto, debiera usarse con gran cuidado.

Véase la sección anterior y el Apéndice 4 para más detalles.

### 2.3.5.4 INLINE(C1,C2,C3,.....)

Este procedimiento permite insertar un programa en código máquina del Z80 **inlínea** con el programa en Pascal. Los valores (C1 MOD 256, C2 MOD 256, C3 MOD 256) se cargan dentro del código objeto generado por el compilador a partir de la dirección actual del contador que el compilador maneja. C1, C2, C3, etc., son constantes enteras y puede haber cualquier cantidad de ellas. Consulta el Apéndice 4 para un ejemplo del uso del procedimiento INLINE.

### 2.3.5.5 USER(V)

USER es un procedimiento para **ceder** el control al **usuario,** y tiene un solo parámetro **V** que es entero. Este procedimiento hace que se **cite** -y ponga en marcha- la rutina situada a partir de la dirección de memoria dada por **V.** Dado que Hisoft Pascal representa a los enteros en la forma de **complemento a doses** (véase Apéndice 3) con el fin de referirse a direcciones mayores de #7FFF (32767), deben usarse valores negativos para **V.** Por ejemplo #9ØØØ es -28672 y por tanto USER(-28672); provocará una **llamada** a esa dirección de memoria. Sin embargo, al usar una constante para referirse a una dirección de memoria, es mucho más conveniente usar la notación hexadecimal.

La rutina **citada** -puesta en marcha- deberá terminarse con la instrucción de **vuelta** del Z80 (RET#C9) y debe preservar el registro IX.

Los registros A, B, C, D, E, H, L y F son cargados a partir de los valores RA, RB, RC, RD, RE, RH, RL y RF respectivamente antes de ceder el control a la rutina.

Los valores devueltos por la rutina en los registros, se entregan como valores de las variables mencionadas. Véase Sección 2.4.2.

### 2.3.5.6 HALT

Este procedimiento de **detención** hace que se pare la ejecución del programa con el mensaje 'Halt at PC=XXXX' indicando que está detenido en contador de programa = XXXX, estando expresada la dirección de memoria XXXX en hexadecimal. Además del listado de la compilación, se puede usar HALT para determinar cuál de dos o más caminos a través del programa ha sido tomado. Esto se usará normalmente sólo durante la **depuración** del programa.

### 2.3.5.7 POKE(X,V)

El procedimiento POKE sirve para **meter,** o depositar el valor de la expresión **V** a partir de la dirección de memoria dada por **X.** Obviamente **X** es del tipo entero, y **V** puede ser cualquier tipo exceptuando el **grupo** (SET). Véase la Sección 2.3.5.5 para la explicación de cómo se usan los enteros para representar las direcciones de memoria. Ejemplos:

```
POKE(#6ØØØ,'A')        mete #41 en la dirección #6ØØØ
POKE(-28672,3.6E3)      mete ØØ ØB 8Ø 7Ø en la dirección #9ØØØ.
```

### 2.3.5.8 TOUT(NAME, START, SIZE)

TOUT es el procedimiento que se usa para **ex**pedir variables hacia la **cinta** (tape). El primer parámetro **nombre** es del tipo ARRAY[1..12] OF CHAR y es el nombre con que quedará registrado el fichero en la cinta. Se vierte a la cinta el contenido de un bloque de memoria de longitud dada por el parámetro **tamaño,** a partir de la dirección dada por el parámetro **comienzo.** Estos dos parámetros SIZE y START son del tipo INTEGER.

E.g. para guardar en cinta el valor de la variable **V,** bajo el nombre 'VAR' usa:

```
TOUT('VAR_________',ADDR(V),SIZE(V))
```

El uso de direcciones **reales** de memoria, da al usuario mucha más flexibilidad que la mera posibilidad de guardar variables múltiples. Por ejemplo, todas las variables **globales** de un programa pueden guardarse en un fichero separado. Véase el Apéndice 4 para un ejemplo del uso de TOUT para **ex**pedir datos a la **cinta.**

### 2.3.5.9 TIN(NAME,START)

Este procedimiento se usa para **in**gresar desde la **cinta** (tape) información que previamente se ha guardado usando el procedimiento TOUT. El parámetro **nombre** es del tipo ARRAY[1..12] OF CHAR y el parámetro **comienzo** es del tipo INTEGER. Se busca en la cinta el fichero dado por el parámetro NAME y cuando se encuentra se carga a partir de la dirección de memoria dada por el parámetro START. El número de bytes a traer de la cinta y cargar en la memoria, se toma de la propia información de la cinta (ya que lo graba automáticamente el procedimiento TOUT).

E.g. para traer a la memoria la variable guardada en el ejemplo de la Sección 2.3.5.8, debes usar:

```
TIN('VAR_________  ',ADDR(V))
```

Véase Apéndice 4 para un ejemplo del uso de TIN para ingresar datos desde la cinta.

### 2.3.5.10 OUT(P,C)

Este procedimiento se utiliza para **accesar** directamente los "portales de salida" del Z80 sin usar el procedimiento INLINE. El valor del parámetro entero P se carga en el registro BC del microprocesador, y el parámetro **literal** C se carga en el registro A, y a continuación se ejecuta la instrucción de ensamblador OUT(C),A para llevarlo fuera.

E.g. OUT(1,'A') deposita el carácter 'A' en el portal 1 de salida del Z80.

### 2.3.5.11 EXTERNAL(S1,V1,V2,...)

Este procedimiento permite **ceder** la ejecución de una tarea a programas **externos** grabados en ROM o que han de ser implantados en la memoria de escritura-lectura (llamados RSX=Resident System Extension) desde un almacenamiento externo, de manera similar al comando **barra** '|' del BASIC. El primer parámetro es un **literal** que corresponde al nombre identificativo del **comandos** externo. Puede ir seguido de cualquier número de parámetros, sean del tipo entero, o literal de uno o varios caracteres. Para transpasar una variable **V** a un comando externo, usa ADDR(V) para indicarle su **dirección.** Las minúsculas en el nombre del comando se convierten a mayúsculas. Los **descriptores** de los valores literales están adjudicados en el **cúmulo** que maneja el Pascal para cada procedimiento.

Ejemplo: EXTERNAL('BASIC');
efectúa un regreso al BASIC de una manera bien controlada.

El uso principal de este procedimiento es para conseguir acceso al sistema de discos. Por ejemplo:

```
EXTERNAL('DIR','*.COM');
```

dará un **directorio** de los ficheros de la clase .COM en la unidad de disco actual.

Si deseas usar un RSX deberás **presentárselo** al programa gestor grabado en ROM del sistema, usando la rutina KL LOG EXT al comienzo de tu programa. Eso es necesario porque cuando un programa finaliza limpia todas las secuencias de sucesos y tiene como efecto colateral la anulación del RSX. Véase el CPC464 Firmware Manual para detalles concernientes a los RSX.

### 2.3.5.12 AFTER(COUNT,TIMER,PROC)

AFTER corresponde al comando BASIC del mismo nombre, usado para suscitar la ejecución de una rutina **después** de transcurrir un determinado lapso de tiempo.

El primer parámetro, el de **cuenta**, es un entero que representa la cantidad de intervalos de 1/50-avo de segundo, transcurrido el cual se citará -se pondrá en marcha- el **procedimiento** mencionado como tercer parámetro del comando, y que no permite parámetros propios. El segundo parámetro es el **temporizador** a usar. Debe ser un entero entre 0 y 3, ambos inclusive, para indicar cuál de los "cronometradores" posibles es el que se va a emplear.

e.g. AFTER(1ØØ,1,FRED); hará que se desvíe el curso de ejecución para ejecutar el procedimiento llamado FRED **después** de 2 segundos, cronometrados por el temporizador 1.

### 2.3.5.13 EVERY(COUNT,TIMER,PROC)

EVERY corresponde al comando BASIC del mismo nombre, que suscita la ejecución de una rutina **cada** vez que ha transcurrido un determinado lapso de tiempo. Sus parámetros son los mismos que para EVERY (véase Sección 2.3.5.12). Véase el manual del BASIC para más detalles sobre los temporizadores.

e.g. EVERY (3ØØ,2,FRED); hará que se desvíe el curso del programa para ejecutar el procedimiento denominado FRED **cada** 6 segundos.
El procedimiento será añadido con prioridad 2 a la secuencia de instancias de interrupción que están esperando ser atendidas por el procesador.

### 2.3.5.14 SOUND(G,K,L,H,M,J,I)

El procedimiento SOUND para hacer que **suene** una determinada nota, de acuerdo con los 7 parámetros enteros siguientes:

1. Canales a usar y requisitos de **acorde** (estado del canal).
2. Envolvente de volumen a usar para la nota.
3. Envolvente de tono a usar.
4. Período de tono.
5. Período de ruido.
6. Amplitud inicial.
7. Duración o cantidad de repeticiones de la envolvente.

Estos parámetros corresponden a los equivalentes en el comando SOUND del BASIC, pero observa que están en un orden diferente y que deben especificarse **todos.**

Cuando se termina un programa se detienen todos los sonidos. El programa de sonido queda registrado en el **cúmulo** del procedimiento pertinente. Véase el Capítulo 6 de la guía del usuario del BASIC para más detalles.

### 2.3.5.15 ENV(N,P1,Q1,R1,P2,Q2,R2,...)

El comando ENV para fijar la **envolvente de volumen** de una nota sonora corresponde al comando en BASIC del mismo nombre, y admite parámetros enteros. Su primer parámetro es el número identificativo de la envolvente (1..15). Y luego siguen hasta 5 **secciones** de envolvente que pueden ser controladas a través del hardware o del software, con cada sección conteniendo hasta 3 parámetros enteros.

Las secciones de envolvente de software tienen los siguientes componentes:

1. Cuenta de pasos en la sección
2. Altura del paso en la sección
3. Lapso, o duración de paso, en la sección

Las secciones de hardware tienen los siguientes componentes:

1. Forma de envolvente (Mayor de 128). El valor de este parámetro menos 128 es el enviado al registro 13 del generador programable de sonidos.
2. Byte bajo del período de envolvente que se envía al registro 11.
3. Byte alto del período de envolvente que se envía al registro 12.

2.3.5.16 ENT(S,T1,V1,W1,T2,V2,W2,...)

El comando ENT para fijar la **envolvente de tono** de una nota, corresponde al comando BASIC del mismo nombre, y admite parámetros enteros. El primer parámetro es el número identificativo de envolvente (-15..15). Si el número de envolvente es negativo, implica que se repita un determinado número de veces, al igual que en el comando en BASIC. Luego siguen hasta cinco secciones de envolvente. Para lo relacionado con estas secciones, los parámetros son los mismos que en el comando en BASIC. Luegos siguen hasta 5 secciones de envolvente. Para las secciones **relativas** de envolvente de tono, la forma es la misma que para el comando en BASIC.

La sección **absoluta** correspondiente a:

```
=toneperiod,pausetime    (2parameters)
```

que fija el período de tono y la duración del **lapso** en BASIC, es equivalente en Pascal a:

```
240+toneperiod DIV 256, toneperiod MOD 256,
   pausetime       (3parameters)
```

es el mismo formato usado por el programa **grabado** en la memoria de únicamente escritura.

Consulta el Capítulo 8 de la guía del Usuario BASIC para más detalles sobre los comandos ENT y ENV.

## 2.3.6 Funciones Predefinidas Adicionales

2.3.6.1 RANDOM(X)

RANDOM genera un número **pseudoaleatorio** en la banda 0 - MAXINT, i.e. un **entero** positivo. RANDOM acepta un parámetro y si es cero el resultado es el siguiente número aleatorio de la secuencia; en los demás casos el parámetro se toma como **germen** para una nueva secuencia de números aleatorios.

2.3.6.2 SUCC(X)

**X** puede ser cualquier dato de tipo **scalar** exceptuando el tipo real y SUCC(X) entrega el **sucesor** de **X.** Ejemplos:

```
SUCC('A')   devuelve 'B'        SUCC('5')  devuelve '6'
```

2.3.6.3 PRED(X)

**X** puede ser cualquier dato de tipo scalar exceptuando el tipo real. El resultado de la función es el **predecesor** de **X.** Ejemplos:

```
PRED('j')  entrega  'i'        PRED(TRUE) entrega FALSE
```

2.3.6.4 ODD(X)

**X** debe ser del tipo entero. La función ODD (impar) da un resultado Booleano que es TRUE (=cierto) si **X** es **impar** y el resultado FALSE (=falso) si **X** es **par.**

2.3.6.6 ADDR(V)

Esta función acepta como parámetro el identificador de una variable de cualquier tipo, y entrega como resultado un **entero** que es la **dirección** (=address) de memoria donde está alojado el valor de la variable **V.** Para saber cómo se guardan las variables en memoria, durante la ejecución y en el Hisoft Pascal, véase el Apéndice 3. Para un ejemplo del uso de ADDR véase Apéndice 4.

2.3.6.7 PEEK(X,T)

El primer parámetro de esta función es del tipo entero y especifica la **dirección** de la memoria cuyo contenido se desea **mirar** -examinar-. El segundo argumento es un **tipo,** y es el tipo que tendrá el resultado de la función.

PEEK se usa para obtener datos de la memoria del ordenador y el resultado puede ser de cualquier tipo.

En todas las operaciones de PEEK (mirar) y POKE (meter) con la memoria, los datos se mueven de acuerdo con la representación interna propia del Hisoft Pascal detallada en el Apéndice 3. Por ejemplo, si a partir de la dirección de memoria #9ØØØ están **contenidos** los valores: 5Ø 61 73 63 61 6C (dados en notación hexadecimal) entonces:

```
WRITE(PEEK(#9ØØØ,ARRAY[1..6] OF CHAR))  da 'Pascal'
WRITE(PEEK(#9ØØØ,CHAR))                 da 'P'
WRITE(PEEK(#9ØØØ,INTEGER))              da 24912
WRITE(PEEK(#9ØØØ,REAL))                 da 2.46227E+29
```

Véase el Apéndice 3 para más detalles sobre las representaciones de datos dentro del Hisoft Pascal.

2.3.6.7 SIZE(V)

El parámetro de esta función de **tamaño** es una variable. El resultado es un entero que refleja la cantidad de almacenamiento ocupado por esa variable en bytes.

#### 2.3.6.8 INP(P)

INP se utiliza para **accesar** directamente un portal de entrada del Z80, sin tener que usar el procedimiento INLINE. El valor del parámetro entero **P** se carga en el registro **BC** y el **carácter** resultante de aplicar la función se consigue ejecutando la instrucción de lenguaje ensamblador IN A,(C)para **introducir** el dato que se haya depositado en el portal de entrada señalado.

#### 2.3.6.9 INITEVENT(CLASS,PROC)

Esta función **inicializa** (init) un bloque de **eventos** (event) y entrega como resultado un entero que representa la dirección correspondiente a ese evento. Su primer parámetro es la **clase** del evento, tal y como se detalla en el CPC464 Firmware Manual. El evento debe ser síncrono. El segundo parámetro es un **procedimiento** -que no tiene parámetros propios- y que es el **citado** cuando ocurre el evento en cuestión. El "bloque de eventos" está incluido en el **cúmulo** de variables que el programa maneja. Esta función puede ser usada por el programador avanzado para conseguir acceso a las potentes facetas del **núcleo** del sistema operativo, que no están habitualmente disponibles en lenguajes de alto nivel.

#### 2.3.6.10 SQ(CHANNEL)

Esta función, para ver si hay notas en la secuencia de sonidos a emitir por un determinado **canal,** corresponde a la función del mismo nombre en BASIC y entrega como resultado un entero que refleja el estado de la secuencia de notas a emitir por el canal dado por el argúmento de la función. Véase el manual del BASIC para más detalles. Por ejemplo:

```
WRITE(SQ(1));
```

dará 4 si no se ha dado ningún comando SOUND para el canal A de sonido.

#### 2.3.6.11 REMAIN(TIMER)

Es una función con un argumento entero, y que corresponde a la función del mismo nombre en BASIC, usada para saber cuánto tiempo **queda** o **resta** del intervalo de tiempo estipulado para un determinado temporizador. El resultado es un entero y el argumento refleja cuál de los cuatro posibles temporizadores. Al aplicar esta función, se **cancela** el temporizador en cuestión.

## 2.4 Variables predefinidas

### 2.4.1 ERRFLG y ERRCHK

ERRFLG y ERRCHK son variables Booleanas que se usan para detectar **errores** cuando el usuario impone datos por el teclado, que deben ser numéricos. Si ERRCHK es **cierto,** es porque ha ocurrido un error de **dígito esperado,** y en lugar de detener la ejecución del programa se alza el **testigo** ERRFLG poniéndolo al valor cierto, y se impone un cero como valor tecleado. Normalmente ERRFLG es falso.

Por ejemplo:

```
ERRCHK:=TRUE;
REPEAT
 READLN; READ(I)
 IF ERRGLF THEN WRITE('Por favor teclea un número')
UNTIL NOT ERRFLG;
```

### 2.4.2 RA,RB,RC,RD,RE,RH,RL y RF

Estas variables del tipo **literal,** o carácter se usan en unión del procedimiento USER (véase Sección 2.3.5.5). Sus valores se usan para dar los valores iniciales de los registros del Z80 antes de ceder el control a una rutina de **usuario,** y se actualizan con los valores pertinentes entregados por dicha rutina. Por ejemplo:

```
RA:='a'; USER(#BB5A)
```

sacará el carácter 'a' vía la rutina TXT OUTPUT del sistema operativo, en la dirección #BB5A.

Después de **activar** el procedimiento mediante:

```
USER(#BB06);
```

la variable RA contendrá el valor que el sistema operativo devuelve después de efectuar la rutina KM WAIT CHAR.
Las variables predefinidas RAF,RBC,RDE y RHL se modifican acordemente cuando cambiar los valores de las variables RA,RB, etc.

### 2.4.3 RAF,RBC,RDE y RHL

Estas variables del tipo entero se usan en conjunción con el procedimiento USER (véase Sección 2.3.5.5). Sus valores se usan para establecer los valores iniciales de los registros del Z80 antes de citar una rutina de **usuario,** y se actualizan con los valores correspondientes que entrega la rutina como resultados. Por ejemplo:

```
RA:=CHR(I);
USER(#BCC5); (SOUND T ADDRESS)
IF ODD(RAF) THEN
      WRITE('Dirección de envolvente tono',RHL:4:H)
 ELSE WRITE('Envolvente no hallada')
```

mostrará la dirección de la envolvente **I,** si existe. Observa el uso de la función ODD(RAF) que determina si es **impar** el contenido de los registros **solidarios** AF del Z80, y por tanto será **cierta** si el **testigo de acarreo** está alzado.

Cambiando valores en RHL, etc. modificará los valores correspondientes de RH, RL, etc. (Véase Sección 2.4.2).

# SECCION 3
# COMENTARIOS Y OPCIONES DEL COMPILADOR

## 3.1 Comentarios

Un comentario puede aparecer entre dos palabras reservadas cualesquiera, números, identificadores o símbolos especiales -véase Apéndice 2. Un comentario empieza con un carácter 'llave de apertura ('{'), o con la pareja de caracteres '**(***'. A no ser que el siguiente carácter sea el signo dólar '**$**', todos los caracteres serán **ignorados** hasta que aparezca la siguiente llave de cierre '}' o la pareja de caracteres '***)**'. Si se encuentra un signo **$**, entonces el compilador buscará una serie de opciones (véase más adelante) para saber los caracteres que debe saltarse hasta que se encuentre una '}' o una pareja de '***)**'. Las llaves '{' y '}' pueden obtenerse en el teclado usando la tecla de **turno** marcada **SHIFT** y uno u otro de los corchetes '[' y ']', respectivamente.

## 3.2 Opciones del Compilador

Las opciones del compilador se incluyen en el programa entre **marcas de comentario,** y la primera opción en la lista va **precedida** de un símbolo '**$**'.

Ejemplo:

`(*$C-,A-*)` para quitar las comprobaciones de teclado y de tablas.

Las letras **clave** de las opciones del compilador pueden estar en mayúsculas o en minúsculas. Se dispone de las siguientes opciones:

`Opción L:`

Controla el listado del programa y las direcciones en código objeto.

Si se usa `L+` se dará un listado completo.
Si se usa `L-` sólo se listarán las líneas donde se haya detectado un error.

Prescrito para omisiones: `L+`

`Opción O:`

Controla si se van a efectuar comprobaciones de **rebase** ('overflow') al tratar con números. Normalmente siempre se comprueban la multiplicación y la división de enteros y todas las operaciones aritméticas con números reales, para ver si **desbordan** el número de cifras permitidas.

Si se usa `O+` se harán las comprobaciones sobre la suma y la resta con enteros.
Si se usa `O-` no se harán las comprobaciones mencionadas.

Prescrito para omisiones: `O+`

Opción C:

Controla si van a efectuarse **comprobaciones** (checks) durante la ejecución del programa en código objeto, sobre lo que sucede en el teclado. Si se usa C+, pulsando cualquier tecla detendrá temporalmente la ejecución del programa; la pulsación subsiguiente de [**ESC**]ape hará que se abandone la ejecución con un mensaje de HALT (véase Sección 2.3.5.6), y la de cualquier otra tecla hará que siga la ejecución.

Esta **comprobación** se hace al comienzo de todos los **bucles,** procedimientos y funciones. Por tanto el usuario puede emplear esta posibilidad para detectar qué bucle, etc., no ha terminado correctamente durante el proceso de **depuración.** Ciertamente deberá estar **inhibida** la comprobación, si deseas que el programa objeto se ejecute rápidamente.

El uso de la opción C+ no incrementa el tamaño del programa objeto. Si se usa C- no se efectuará la comprobación mencionada.

Prescrito para omisiones: C+

Opción S:

Controla si se han de efectuar o no comprobaciones del espacio ocupado por la **percha** donde "cuelga" (stack=apilar ordenadamente) los parámetros que intervienen en los procedimientos y funciones.

Si se especifica S+ al comienzo de cada ejecución de un procedimiento o de una función que se cite en el programa, se hará una comprobación para ver si con la **percha** correspondiente se sobrepasará en ese bloque el espacio disponible. Si la percha usada en la ejecución se "desborda" e intenta ocupar el **cúmulo** donde se "apilan" las variables dinámicas, o el área que contiene al programa, se mostrará el mensaje 'Out of RAM at PC=XXXX' para indicar que se queda **sin memoria de escritura-lectura en PC=XXXX,** y se abandona la ejecución de ese programa. Naturalmente, estas comprobaciones no son **infalibles;** y si un procedimiento utiliza en grado sumo la **percha** disponible, dentro de sí mismo, el programa puede 'bloquearse'. Alternativamente, si un procedimiento utiliza muy poca cantidad de memoria como percha, mientras que sigue aprovechando la **recursión** (citándose a sí mismo), es posible que dicho procedimiento se vea detenido innecesariamente por las comprobaciones efectuadas.

Si se especifica como opción S- no se efectúa ninguna de las comprobaciones mencionadas.

Prescrito para omisiones: S+

Opción A:

Controla si se han de efectuar comprobaciones para asegurarse que los **índices** empleados para señalar los elementos de las variables múltiples (array) están dentro de las cotas especificadas al declararla.

Si se especifica A+ y un **índice** es demasiado alto o demasiado bajo, se mostrará el mensaje 'Index too high' o el 'Index too low' según corresponda, y se detendrá la ejecución del programa.

Si se especifica como opción A- no se efectuarán tales comprobaciones.

Prescrito para omisiones: A+

Opción I:

Cuando se emplea aritmética entera con notación de complemento a doses, y operando con 16 bits ocurre un **rebase** cuando al efectuar una comparación >, <, >= o <=, los argumentos difieren en una cantidad mayor a MAXINT (32767). Si eso ocurre, el resultado de la comparación será incorrecta. Normalmente, esto no presenta ninguna dificultad; sin embargo, si el usuario desea comparar tales números **enteros** (integers) el uso de la opción I+ asegura que el resultado de tales comparaciones será siempre correcto. La situación equivalente puede surgir con la aritmética de números reales en cuyo caso se lanzará un error de rebase, si los argumentos difieren en una cantidad mayor de 3.4E38. Y eso no puede evitarse.

Si se especifica I- no se efectuará ninguna comprobación en las operaciones mencionadas.

Prescrito para omisiones: I-

Opción P:

Si se usa la opción P se puede cambiar el equipo periférico hacia el que se envíe el listado de la compilación, i.e. si se estaba usando la pantalla de video, se pasa a usar la impresora, y viceversa. Observa que esta opción no va seguida de un signo más ni de un signo menos.

Prescrito para omisiones: Se usa la pantalla de video, no la impresora.

Opción F:

Esta letra clave de opción debe ir seguida de un espacio y luego de un **nombre de fichero** de 12 caracteres. Si el nombre del fichero tiene menos de 12 caracteres, se rellenará automáticamente con espacios en blanco por la derecha.

La presencia de esta opción hace que se incluya un **programa fuente** en Pascal tomado del fichero especificado y a partir del final de la línea actual. Es muy útil si el programador desea construir una "programoteca" en cinta con los procedimientos y funciones más frecuentemente usados, y luego incluirlos en programas particulares.

Los programas pueden **guardarse** en la cinta, usando el comando intrínseco del editor de clave 'P'.

Por ejemplo:

```
(*$F MATRIX      incluye el texto de un fichero en cinta llamado
                 MATRIX*)
```

Cuando se escriben programas muy largos, puede que no haya suficiente sitio en la memoria para que estén presentes al mismo tiempo el programa **fuente** y el **objeto.**

Sin embargo, es posible aún compilar tales programas, guardándolos en cinta y usando esta opción de clave 'F' (en ese caso, en cualquier momento sólo hay en la RAM de la máquina 128 bytes del programa fuente, dejando mucho más sitio para el código objeto).

Esta opción no puede **anidarse** (incluirse dentro de la misma opción).

Las opciones del compilador pueden usarse selectivamente. Así se pueden acelerar y condensar secciones de código objeto ya **depuradas,** quitando las comprobaciones incluidas, mientras que se mantienen en otros trozos de código que todavía no se ha probado.

# SECCION 4
# EL EDITOR INTEGRAL

## 4.1 Introducción al Editor

El **editor** suministrado con todas las versiones del Hisoft Pascal es un editor simple, basado en la **línea,** y diseñado para trabajar con todos los sistemas operativos del Z80, manteniendo la facilidad de uso y la habilidad de revisar y corregir programas rápida y eficazmente. Este editor ha sido mejorado para el AMSTRAD CPC464 añadiéndole posibilidades de editor de **pantalla** mediante el uso de la tecla COPY junto con los cursores y la tecla de **turno** (marcada **SHIFT**). Estas facultades extra se explican en detalle a continuación.

El **texto** del programa se conserva en memoria en una forma compacta; el número de espacios delanteros en una línea es sólo de un carácter al comienzo de la línea, y todas las **palabras reservadas** están escritas **taquigráficamente,** ocupando un solo carácter (algunos los llaman "**glifos**"). Esto comporta una reducción en el tamaño del texto del 25%, típicamente.

Se **"entra"** automáticamente (a trabajar con) el editor cuando se carga en memoria desde la cinta el Hisoft Pascal, y aparece el mensaje de saludo:

```
HiSoft Pascal Amstrad CPC464
Version of 26/9/84
Copyright Hisoft 1983,4
All rights reserved
```

y luego una pantalla de ayuda, seguida de la **divisa** del editor, que es el símbolo '>'.

Como respuesta a esta presentación, puedes teclear una línea de comando con el siguiente formato:

`CN1,N2,S1,S2` y concluyes el comando pulsando **[ENTER]**

siendo:

| | |
|---|---|
| C | el comando que ha de ejecutarse (véase Sección 4.2). |
| N1 | es un número en la banda 1..32767, ambos inclusive. |
| N2 | es un número en la banda 1..32767, ambos inclusive. |
| S1 | es una **serie** de caracteres con una longitud máxima de 20 caracteres. |
| S2 | es una **serie** de caracteres con una longitud máxima de 20 caracteres. |

La coma se usa para separar los diversos argumentos y parámetros del comando (aunque eso puede cambiar, como por ejemplo en el comando de clave '**Q**') y se ignoran los espacios, exceptuando cuando son parte interna de los propios **literales.** Ninguno de los argumentos es obligatorio, aunque alguno de los comandos (e.g. el comando de clave '**D**' para suprimir) no será cumplimentado si no se especifican N1 y N2. El editor recuerda los números previos y los **literales** que has impuesto por el teclado, y usa esos valores primitivos cuando son aplicables y si no especificas explícitamente un argumento concreto dentro de la línea de comando.

Los valores de N1 y N2 son inicialmente 10 y los literales son inicialmente nulos o vacíos. Si tecleas una línea de comando ilegal, tal como F-1,1ØØ,HELLO, la línea será ignorada, y te mostrará el mensaje 'Pardon?' (¡con el que no te pide perdón! sino que te indica que no lo ha entendido). Puedes entonces volver a teclear la línea correctamente, e.g. F1,1ØØ,HELLO. Este mensaje de error también será el mostrado si la longitud de S2 excede de 20; pero si la longitud de S1 es mayor de 20, entonces todos los caracteres en exceso serán ignorados.

Los comandos pueden escribirse en mayúsculas o en minúsculas.

Mientras se teclea una línea de comando, pueden usarse ciertas funciones de control, e.g. **[CTRL]X** para suprimir todo hasta el comienzo de la línea, **[TAB]** para avanzar el cursor hasta el nuevo tope de tabulación, etc.

Las siguientes sub-secciones detallan los diversos comandos disponibles dentro del editor -observa que siempre que un argumento o parámetro está encerrado entre los símbolos '< >', es una señal de que **debe** estar presente para que pueda cumplimentarse el comando.

## 4.2 Los Comandos para el Editor

### 4.2.1 Inserción de Texto

Se puede insertar texto en el **fichero de texto** bien sea tecleando un número de línea, un espacio y luego el texto requerido, o bien mediante el comando de clave 'I'. Observa que si tecleas un número de línea seguido de **[ENTER]** (i.e. sin ningún texto) dicha línea será suprimida del fichero de texto, si existiera. Siempre que se escribe un texto, las funciones de control **[CTRL]X** (suprimir hasta el comienzo de la línea), **[TAB]** (pasar hasta el siguiente tope de tabulación), **[ESC]** (escapar de la edición) y **[CTRL]P** (bascular impresora/pantalla) pueden emplearse. La tecla **[DEL]** producirá un retroceso **destructivo** (pero no pasa más allá del comienzo de la línea de texto). El texto introducido se "echa en un '**buffer**' interno" primeramente y dentro del tiempo de explotación, y si dicho 'buffer' se llena, se te impedirá que introduzcas ningún texto más -debes usar entonces **[DEL]** o **[CTRL]X** para dejar libre espacio suficiente.

Comando: I n,m

El uso de este comando hace que se pase automáticamente al modo Inserción: se te proporcionan en él números de línea comenzando en **n** e incrementándose en pasos de **m.** Puedes escribir el texto requerido después del número de línea que aparece, usando los diversos códigos de control disponibles si lo deseas, y concluyendo la línea de texto pulsando **[ENTER]**. Para salir de este modo de inserción, usa la tecla de **[ESC]**ape.

Si tecleas una línea con un número de línea que ya existe en el texto, la línea ya existente será **renumerada** con un número mayor en una unidad al que previamente tenía; y la línea que tú tecleas quedará inserta en el texto con el número de línea existente y que has dado, en cuanto concluyas el comando pulsando **[ENTER]**. Si el incremento automático del números de línea produce un número de línea mayor de 32767 se saldrá automáticamente el modo Inserción.

Si al teclear texto, llegas al final de una línea de pantalla sin haber introducido 80 caracteres (el tamaño del 'buffer') entonces se deslizará el texto que haya en pantalla una línea hacia arriba y puedes continuar tecleando en la siguiente línea.

### 4.2.2 Listado del Texto

Puedes dirigir el listado de tu programa bien sea hacia la pantalla del Amstrad (vía el comando '**L**') o hacia la impresora (vía el comando '**Z**').

Comando: **L**n,m

Así se lista el texto vigente mostrándolo en pantalla, a partir del número de línea **n** y hasta el número de línea **m**, ambos inclusive. El valor prescrito si se omite **n**, es siempre 1, y el valor prescrito para **m** es siempre 32767, i.e. observa que los valores prescritos no son los que hubieras estipulado en un comando anterior. Para listar todo el **fichero de texto** con el programa, simplemente usa el comando de clave '**L**' sin ningún parámetro. El listado sacará una **página** (24 líneas) cada vez; y después de exponer una página hará una pausa (si todavía no ha llegado al número de línea **m**) y entonces puedes pulsar **[ESC]** para volver al editor principal, o cualquier otra tecla para que siga el listado.

Comando: **Z**n,m

Lista el fichero de texto por la impresora acoplada. Si no hay conectada ninguna impresora al ordenador, o la impresora está en ese momento **exlínea**, se mostrará el mensaje NO PRINTER! y no se tomará ninguna acción; en los demás casos, el fichero con el texto, entre los números de línea **n** y **m**, ambos inclusive, será imprimido.

Si no se especifican **n** o **m**, se imprimirá el fichero con el texto completo.

Puedes hacer una pausa en el listado, al imprimir, si pulsas cualquier tecla. A continuación, puedes pulsar **[ESC]** para regresar al editor y abandonar el listado por impresora, o cualquier otra tecla para que se reanude la impresión del listado.

### 4.2.3 Edición de Texto

Una vez que hayas creado algún texto, habrá inevitablemente necesidad de **editar** (revisar y corregir) algunas líneas de programa. El editor dispone de diversos comandos que permiten enmendar, suprimir, mover y renumerar líneas. Existen diversas formas de conseguirlo, y se explican a continuación. Algunas de las formas elementales de **edición de pantalla** también están admitidas, y trabajan de la forma siguiente:

Siempre que estás en el modo comando del editor (con la divisa '>' en el margen izquierdo de la línea actual) puedes "separar" el cursor en un **cursor de lectura** y un **cursor de escritura**, si mantienes pulsada la tecla **[SHIFT]** y pulsas una de las teclas de movimiento del cursor.

El cursor de escritura permanecerá en la misma posición en que estaba el cursor original, aunque muevas el cursor de lectura por toda la pantalla (pero no por fuera de la pantalla) usando simultáneamente **[SHIFT]** y las teclas de cursor. Si dejas de presionar **[SHIFT]** y la tecla de cursor pertinente cuando el cursor de lectura esté situado donde tú quieres. Ahora puedes bien teclear directamente el texto que deseas y los caracteres aparecerán en la posición señalada por el cursor de escritura, o puedes pulsar la tecla **[COPY]** y en este caso los caracteres serán transferidos desde la posición del cursor de lectura hasta la posición del cursor de escritura, y las posiciones de ambos cursores serán incrementadas en una unidad.

Para terminar este modo de **copia de pantalla,** simplemente pulsa **[ENTER]:** desaparecerá el cursor de lectura y la línea que contiene el cursor de escritura será examinada normalmente por el editor.

Además de esta capacidad de **edición de pantalla,** también están admitidos diversos comandos para **edición de línea.**

Comando: **D**<n,m>

Se **suprimirán** (delete) todas las líneas de la **n** a la **m,** ambas inclusive, que haya en el fichero de texto. Si **n** < **m,** o no se especifican exactamente los dos argumentos, no se tomará ninguna acción: es para impedir equivocaciones por descuidos. Se puede suprimir una sola línea haciendo **m** = **n;** eso también puede lograrse simplemente tecleando el número de línea seguido de **[ENTER].**

Comando: **M**<n,m,d>

Mueve el bloque de texto entre los números de línea **n** y **m,** ambos inclusive, hasta la posición inmediatamente antes de la línea cuyo número de línea es **d,** y suprime el bloque primitivo de texto. El bloque que se mueve será renumerado comenzando con un número de línea que sea una unidad mayor que el número de línea que precede a la **d.**

No se pueden mover bloques de líneas dentro de ellos mismos, de manera que el parámetro **d** no debe caer dentro del bloque de líneas entre **n** y **m.**

Comando: **N**<n,m>

El uso del comando '**N**' hace que el fichero de texto sea renumerado con el primer número de línea dado por **n,** y en saltos de números de línea dados por **m.** Tanto **n** como **m** deben estar presentes, y si la renumeración hace que cualquier número de línea exceda de 32767, entonces se mantendrá la numeración original.

Comando: **F**n,m,f,s

El texto existente entre la banda de líneas con número de línea x, tal que n < x < m, se escruta, se examina para ver si aparece en él la **serie** de caracteres, el **literal** dado por '**F**' -el literal '**F**' a buscar (find=hallar, encontrar). Si se encuentra dentro del texto dicho literal, se mostrará la línea y automáticamente se entrará en el modo **E**dición.

Puedes usar luego los comandos concernientes al modo edición, para seguir buscando las subsiguientes apariciones del literal '**F**' dentro de la gama de líneas especificada, o sustituirlo por el literal especificado como **s** (sustitutorio) en el comando, y luego continuar la búsqueda de la siguiente aparición de **f**. Véase el modo Edición más adelante para más detalle.

Observa que la banda de líneas donde buscas y que los dos literales que manejas, puede que hayan sido estipulados previamente por cualquier otro comando, de manera que solamente es necesario teclear la clave '**F**' para que se inicie la búsqueda -véase el ejemplo de la Sección 4.3 para mayor clarificación.

Comando: **E**n

Edita la línea con el número de línea dado por **n**. Si dicho número **n** no existe, no se toma ninguna acción; en los demás casos la línea se copia dentro del 'buffer' de entrada y se expone en la pantalla (con el número de línea); debajo de ella se vuelve a mostrar el número de línea en cuestión y se entra automáticamente en el modo Edición. Todo el tratamiento subsiguiente tiene detrás únicamente en el 'buffer' interno y no en el propio fichero de texto que hay en memoria, por lo que la línea original puede recuperarse en el momento que se desee.
En este modo, es conveniente imaginarse un **puntero** que se mueve por toda la línea (comenzando a partir del primer carácter) y que están admitidos diversos subcomandos que te permiten revisar y corregir la línea. Dichos subcomandos son:

**' '** (espacio) -incrementa el puntero del texto en una unidad, i.e. señala hacia el siguiente carácter en la línea. No puedes sobrepasar el extremo final de la línea.

**[DEL]** -decrementa el puntero del texto en una unidad para que señale hacia el carácter anterior en la línea. No puedes retroceder hasta más a la izquierda del primer carácter en la línea.

**[TAB]** -avanza el puntero de texto hasta la siguiente posición de tabulación, pero no puede pasar del extremo final de la línea.

**[ENTER]** -termina la **edición** de esa línea manteniendo todos los cambios hechos en ella.

**Q** -abandona (quit) la edición de esa línea, i.e. **deja** la edición ignorando todos los cambios que se hayan hecho y dejando la línea tal y como estaba antes de que se iniciara su edición.

**R** -recarga el 'buffer' de edición, echando en él otra vez el mismo texto, i.e. olvida todos los cambios hechos en esa línea y vuelve a reponer la línea tal como estaba en un principio.

**L** -lista el resto de la línea que se está editando, i.e. lo que queda de dicha línea a partir de la posición actual del cursor y hasta el final de ella. Permaneces en el modo **edición** con el puntero 'repuntando' hacia el comienzo de la línea.

**K** -elimina (kill) el carácter situado en la posición actual del puntero.

**Z** -suprime todos los caracteres desde (e incluyendo) la posición actual del puntero hasta el extremo final de la línea.

**F** -busca (find=encontrar) la siguiente aparición del **literal a buscar** definido previamente como parámetro '**f**' dentro de una línea de comando. (Véase el comando '**F**'). Este subcomando automáticamente hará que se salga de la edición sobre la línea corriente (conservando los cambios) si no encuentra el literal a buscar dentro de dicha línea actual. Si la aparición del literal a buscar ocurre en una línea posterior (dentro de la banda de líneas previamente especificada), entonces se entrará en el modo edición sobre la línea en que se halla el literal a buscar. Observa que el puntero de texto siempre está situado al comienzo del literal a buscar después de que la búsqueda haya tenido éxito.

**S** -sustituye el literal **sustitutorio,** previamente definido mediante el parámetro **s,** en lugar del literal a buscar que acaba de encontrar, y luego efectúa automáticamente el subcomando '**F**' para buscar la siguiente aparición del literal a buscar. Esto, conjuntamente con el subcomando '**F**' anteriormente explicado, se usa para recorrer un fichero de texto buscando y reemplazando si se desea, un literal dado por otro literal -véase la Sección 4.3 para un ejemplo.

**I** -inserta caracteres en la posición actual del puntero. Permanecerás en este sub-modo hasta que pulses **[ENTER]** -con lo que regresarás al modo principal de edición con el puntero situado después del último carácter que hayas insertado. Usando **[DEL]** dentro de este sub-modo hará que se suprima del 'buffer' el carácter situado a la izquierda del puntero, mientras que el uso de **[TAB]** hará que avance el puntero hasta el siguiente tope de tabulación insertando espacios. Mientras se está en este modo, la forma del cursor es un asterisco '*****'.

**X** -eso avanza el puntero hasta el final de la línea y automáticamente entra en el sub-modo de inserción explicado anteriormente.

**C** -sub-modo de cambio. Eso te permite cambiar el carácter situado en la posición actual del cursor (sobre-escribir o escribir encima) y luego avanza el puntero en una posición. Permaneces en el sub-modo cambio hasta que pulses **[ENTER]**, momento en que regresarás al modo edición con el puntero situado después del último carácter que hayas cambiado. **[DEL]** dentro de este sub-modo simplemente decrementa el puntero en una posición, i.e. lo mueve hacia la izquierda; mientras que **[TAB]** no tiene ningún efecto. Mientras se está en este sub-modo, el cursor adopta la forma del signo más '**+**'.

### 4.2.4 Comandos para Cinta en cassette

El texto se puede guardar en cinta, o traer desde la cinta y cargarlo en memoria, usando los comandos de clave '**P**' y '**G**', y también puede verificarse texto usando el comando de clave '**V**':

Comando: **P**n,m,s

La banda de líneas con números de línea x, tales que n < x < m, se guarda en la cinta registrándola con el nombre de fichero especificado por el parámetro literal **s,** recuerda que estos parámetros pueden tener su valor estipulado previamente por un comando anterior.

Antes de dar este comando, asegúrate que está puesta la lecto-grabadora de cinta y en el modo de grabación.

Comando: **G, , s**

El equipo de almacenamiento -la cinta- es examinada en busca de un fichero de texto cuyo nombre de fichero sea el parámetro literal **s.** Si no se encuentra el fichero solicitado, se mostrará un mensaje de error. Si se encuentra el fichero, será traído de la cinta y cargado en la memoria. Si se detecta un error durante la operación, se mostrará un mensaje de error y se abandonará la operación. Si esto sucede debes **volver a dar** el mismo comando.

Mientras hay una búsqueda en cinta puedes abandonar la operación pulsando la tecla **[ESC].** Eso interrumpirá la transferencia de cinta a memoria y hará que se regrese al editor principal.

Observa que si ya hay algún fichero de texto presente en la memoria, el fichero de texto que se trae desde la cinta quedará **añadido** al fichero existente en memoria, y todo el fichero será automáticamente renumerado comenzando con el número de línea 1 y en incrementos de 1.

Comando: **V, , s**

Verifica un fichero de texto en cinta comparándolo con el existente en memoria.

El equipo de almacenamiento -la cinta- se examina en busca de un fichero cuyo nombre corresponda al dado en el parámetro literal **s.** Cuando se encuentra el fichero se coteja con el fichero de texto actualmente memorizado y sobre el que está trabajando el editor. Si la concordancia es exacta, se muestra el mensaje VERIFIED; en los demás casos se muestra el mensaje FAILED para decir que ha **fallado.**

Comando: **S**n

Estipula la rapidez (speed=velocidad) con que se transfieren los datos desde la memoria a la cinta. Habitualmente, los ficheros en cinta se guardan utilizando la cadencia lenta de 1000 buadios; puedes cambiarlo y usar la cadencia rápida de 2000 baudios, simplemente especificando un número distinto de cero después de la clave **S.** Para volver a utilizar la cadencia lenta, simplemente usa **S** sin ningún número detrás.

E.g. **S**1 para trasvase a cinta a alta velocidad
**S** para trasvase a cinta a velocidad normal.

### 4.2.5 Compilación y Explotación desde el Editor

Comando: **A**

Altera los valores prescritos para omisiones en la compilación/ejecución de programas. Pulsa **A** y concluye con **[ENTER],** y se te pedirá que especifiques el **tamaño de la tabla de símbolo,** mediante el mensaje:

Symbol Table size?

Al que puedes contestar tecleando un número decimal, seguido de **[ENTER]**, para especificar un nuevo valor para el tamaño de dicha tabla. El valor al comienzo de una **sesión** se normalmente 1858, y con ese espacio debe ser suficiente para compilar la mayoría de los programas.

Si simplemente pulsas **[ENTER]** en lugar de imponer un número, el tamaño de la tabla de símbolos no se altera del valor previo. Una vez que hayas tratado con el tamaño de la tabla de símbolos, se te preguntará si deseas o no:

```
Translate Stack?
```

ante lo que puedes responder (especificando un número decimal seguido de **[ENTER]**) la **dirección** en memoria del área que usará -a modo de **percha**- para "apilar ordenadamente" los identificadores que empleará cualquier programa en código objeto que se **traduzca** (translate) de un programa fuente en Pascal, como resultado del comando de clave **'T'** (véase más adelante). Está prescrito que para esta dirección se tome el valor especificado como respuesta a la pregunta sobre la **cima** de la memoria de escritura-lectura que contestaste al "entrar" a trabajar en Pascal.

Descubrirás que es útil fijar la dirección de esta **percha** de traducción, cuando necesitas reservar memoria en la parte superior del espacio de direcciones, para incluir en ella rutinas que deseas interrelacionar con tu programa traducido. Si pulsas **[ENTER]** sin especificar un número, no se cambiará el valor previo del comienzo de dicha **percha** de traducción (y recuerda que "crece hacia abajo, al estar colgada de ella la información").

Comando: **Cn**

Hace que sea compilado el texto que comienza a partir del número de línea dado por el parámetro **n.** Si no especificas un número de línea, se compilará el texto a partir de la primera línea existente. Para detalles adicionales, véase la Sección 0.2.

Comando: **R**

El programa en código objeto obtenido en la compilación previamente mencionada, será ejecutado (run) pero sólo si el programa fuente original no ha sido ampliado en el tiempo transcurrido entre los dos comandos. Véase la Sección 0.2 para más detalles.

Comando: **Tn**

Este es el comando de **'T'**raducción (translate). El programa fuente actual se compila a partir de la línea **n** (o a partir de la primera si se omite **n**), y si la compilación es fructífera, se te pedirá tu confirmación con el mensaje **'Ok?'**; ante el cual, si contestas **'Y'** (Yes=Sí) entonces el código objeto producido por la compilación será movido hasta el final del "explotador" del programa (destruyendo el **compilador**) y luego dicho "programa" y el código objeto obtenido será **vertido** en la cinta, dejándolo registrado como un programa ejecutable -explotable- con un nombre de fichero igual al que esté previamente estipulado para el literal **a buscar**, dado en algún comando anterior por el parámetro **f.**

Este código se vuelca en la cinta en formato de **fichero binario,** de manera que puedes a continuación llevarlo a la memoria desde la cinta, usando el comando LOAD del BASIC CPC464.

El código objeto traducido contiene instrucciones que efectúan una cita a la rutina MC START PROGRAM, para comienzo del código máquina, cuando dicho código se ejecuta; y por tanto, el programa en código objeto toma el control de la máquina, quitando al BASIC y a cualquier RSX que hubiera sido presentado al programa de control en ROM. Por lo tanto, si requieres que esté presente cualquier RSX cuando va a funcionar tu programa en código objeto, debes establecer los valores iniciales por ti mismo dentro de tu código objeto, usando el procedimiento USER del Pascal para **llamar** a la rutina KL LOG EXT (#BCD1) grabada en la ROM, y **presentarle** el RSX pertinente. Observa que el código objeto de la traducción se ubica en (y se mueve hasta) el final de **explotador,** (el programa que actúa en **tiempo de ejecución** de tu programa en Pascal), de manera que después de un comando de '**T**'raducción necesitarás volver a cargar en memoria el compilador. Sin embargo, eso no debe presentar ningún problema ya que sólo es probable que '**T**'raduzcas un programa cuando es completamente operativo.

Si decides no continuar con el vertido en la cinta del programa en código objeto, simplemente teclea un carácter distinto a '**Y**' cuando te lo pide el mensaje 'Ok?': el control de la máquina será devuelto al editor, que continuará todavía funcionando perfectamente dado que el código objeto no se movió de su ubicación en memoria.

### 4.2.6 Otros Comandos

Comando: **H**

Este comando expone una pantalla de ayuda sobre los diversos comandos de edición disponibles. Los comandos se dan en letras mayúsculas.

Comando: **Q**, , d

Este comando te permite que cambies el símbolo **delimitador,** que es el que se usa para separar parámetros y argumentos dentro de una línea de comando. Al principio, para el editor dicho símbolo delimitador es la coma ','; pero puede cambiarse mediante el comando de clave '**Q**' para que sea el primer carácter de la serie especificada por el parámetro **d** de este comando. Recuerda que una vez que has definido un nuevo delimitador, debes usarlo (incluso para el propio comando '**Q**') hasta que se especifique algún otro.

Observa que los símbolos separadores de argumentos no pueden ser espacios.

Comando: **U**

Simplemente muestra el último número de línea de las que hay en el fichero de texto actual. Este comando es útil para encontrar el final del fichero de texto que estás editando, de manera que le puedas añadir fácilmente líneas de comando o que puedas listar el final del fichero.

Comando: **W**

Bascula (pone y quita) el modo de pantalla entre 40 caracteres por línea y 80 caracteres por línea. Inicialmente está prescrito que la pantalla esté en el modo 1, i.e. 40 caracteres por línea. Lo cambias a 80 por línea, usando **W [ENTER]**, una vez, y luego puedes cambiarlo de nuevo a 40 caracteres por línea pulsando **W [ENTER]** por segunda vez.

Comando: **Y**

Este comando no acepta parámetros y simplemente muestra los valores actuales prescritos para el símbolo delimitador, la banda de líneas **(n-m)** y los literales a buscar **'f'** y sustitutorio **'s'**. Debe recordarse que ciertos comandos al editor (como **'D'** y **'L'**) no usan la banda de líneas prescrita para omisiones, sino que han de especificarse explícitamente los valores en la línea de comando.

Comando: **X**

Este comando muestra las direcciones del comienzo y el final del texto en notación hexadecimal. Es útil para calcular el tamaño de tu fichero de texto en bytes.

Comando: |

El comando **barra** vertical '|' te permite recurrir a comandos de **retaguardia,** grabados en la ROM, desde dentro del editor.

La barra debe ir seguida de un nombre de comando válido para el programa en la ROM externa que la va a ejecutar, y opcionalmente puede ir seguido de cualquier cantidad de parámetros que necesite dicho comando. Los parámetros deben estar separados por una coma, y cuando son **literales** deben encerrarse entre **apóstrofes** ('), y no entre comillas (").

Si el comando o los parámetros traspasados a las rutinas externas, no son válidos, se mostrará el mensaje de error `'Pardon'` (que es una manera elegante de decir que no lo entiende).

Por ejemplo: `|dir,'*.PAS'`

permite obtener el directorio de un disco, con la lista de todos los ficheros que tengan como clase de fichero la de .PAS. Y por tanto:

```
|basic
```

permite regresar a trabajar con el BASIC.

## 4.3 Un Ejemplo del uso del Editor

Supongamos que has tecleado el siguiente programa (usando I10,10):

```
1Ø  PROGRAM BUBBLESORT
2Ø  CONST
3Ø  Size = 2ØØØ;
4Ø  VAR
5Ø  Numbers : ARRAY [1..Size] OF INTEGER;
6Ø  I, Temp : INTEGER;
7Ø  BEGIN
8Ø  FOR I:=1 TO Size DO Number[I] :=RANDOM;
9Ø  REPEAT
1ØØ FOR I:=1 TO Size DO
11Ø Noswaps := TRUE;
12Ø IF Number[I] > Number[I+1] THEN
13Ø BEGIN
14Ø Temp := Number[I];
15Ø Number[I] := Number[I+1];
16Ø Number[I+1] := Temp;
17Ø Noswaps := FALSE
18Ø END
19Ø UNTIL Noswapss;
195 FOR I := 1 TO Size DO WRITE(Number[I]:4)
2ØØ END.
```

Que como puedes ver efectúa un **ordenamiento** (sort) de datos según el método llamado **de burbuja** (bubble), recorriendo la "hilera" de datos a ordenar hasta que en una pasada no haya **ningún canje** (no swap) de posiciones de los elementos. El programa tiene los errores siguientes:

| | |
|---|---|
| Línea 10 | Falta el punto-y-coma. |
| Línea 30 | No es realmente un error, pero queríamos un **tamaño** de 100. |
| Línea 100 | El valor de finalización del bucle debe ser 'Size-1'¡ |
| Línea 110 | Debiera estar en la línea 95 en lugar de estar aquí. |
| Línea 190 | No es extraño que hayamos deletreado mal ese identificador tan raro de variable. |

Además, la variable Numbers ha sido declarada, pero siempre se menciona como Number. Finalmente la variable BOOLEAN Noswaps no ha sido previamente declarada.

Para hacer que todo sea correcto, podemos proceder como sigue:

| | |
|---|---|
| F6Ø,21Ø,Number,Numbers | y luego usar repetidamente el subcomando **'S'** |
| E1Ø | luego sucesivamente los comandos **X;[ENTER] [ENTER]** |
| E3Ø | luego _____K C 1 **[ENTER][ENTER]** |
| F1ØØ,1ØØ,Size,Sixe-1 | seguido del subcomando de sustitución |
| M11Ø,95 | luego **X[DEL][ENTER][ENTER]** |
| E19Ø | |
| 65Noswaps:BOOLEAN; | |
| N1Ø,1Ø | para renumerar el programa con saltos de 10. |

Te recomendamos que practiques en el ejemplo anterior usando **realmente** el editor de Pascal.

# APENDICE 1
# ERRORES

### A.1.1 Números de error generados por el compilador (*)

1. Número demasiado grande
2. Esperado un punto-y-coma.
3. Identificador sin declarar.
4. Esperado identificador.
5. Usa '=' no ':=' en una declaración de constante.
6. Esperado '='.
7. Este identificador no puede comenzar una sentencia.
8. Esperado ':='.
9. Esperado ')'.
10. Tipo erróneo.
11. Esperado '.'.
12. Esperado factor.
13. Esperada constante.
14. Este identificador no es una constante.
15. Esperado 'THEN'.
16. Esperado 'DO'.
17. Esperado 'TO' o 'DOWNTO'.
18. Esperado '('.
19. No puede escribir este tipo de expresión.
20. Esperado 'OF'.
21. Esperado ','.
22. Esperado ':'.
23. Esperado 'PROGRAM'.
24. Esperada variable dado que el parámetro es un parámetro variable.
25. Esperado 'BEGIN'.
26. Esperada variable en la cita a READ.
27. No puede comparar expresiones de este tipo.
28. Debe ser bien tipo INTEGER o tipo REAL.
29. No puede READ este tipo de variable.
30. Este identificador no es un tipo.
31. Esperado exponente en número real.
32. Esperada expresión scalar (no numérica).
33. No permitidas las cadenas de literales nulas (usa CHR(0)).
34. Esperado '['.
35. Esperado ']'.
36. El tipo del índice de array debe ser scalar.
37. Esperado '..'.
38. Esperado en declaración ARRAY un ']' o un ','.
39. Cota inferior mayor que cota superior.
40. Grupo demasiado grande (más de 256 elementos posibles).

**(*)** En las páginas A1.2 y A1.3 puedes ver los números de error en inglés

41. Resultado de función debe ser del tipo identificador.
42. Esperado en el grupo un ',' o un ']'.
43. '..' o ',' o ']' esperado en un grupo.
44. Tipo de parámetro debe ser un identificador de tipo.
45. El grupo nulo no puede ser el primer factor en una sentencia de no-asignación.
46. Esperado scalar (incluyendo real).
47. Esperado scalar (no incluyendo real).
48. Grupos incompatibles.
49. '<' y '>' no puede usarse para comparar grupos.
50. 'FORWARD', 'LABEL', 'CONST', 'VAR', 'TYPE' o 'BEGIN' es el identificador esperado aquí.
51. Esperada cifra hexadecimal.
52. No puedo POKE grupos.
53. Array demasiado grande (>64K).
54. 'END' o ';' esperado en la definición de ficha.
55. Esperado identificador de campo.
56. Esperada variable después de 'WITH'.
57. La variable en WITH debe ser del tipo RECORD.
58. El identificador de campo no tiene asociada sentencia WITH.
59. Esperado entero sin-signo después de'LABEL'.
60. Esperado entero sin-signo despuésde 'GOTO'.
61. Esta etiqueta está a nivel erróneo.
62. Etiqueta no declarada.
63. El parámetro de SIZE debe ser una variable.
64. Sólo puede usarse pruebas de igualdad para punteros.
65. El único parámetro de escritura para enteros con dos ':'s es el de forma e:m:H.
68. Las series de caracteres no pueden contener caracteres de fin de línea.
69. El parámetro de NEW, MARK o RELEASE debiera ser una variable de tipo puntero.
70. El parámetro de ADDR debe ser una variable.
71. Este parámetro debe ser un procedimiento.
72. Este parámetro debe ser un procedimiento sin parámetros.
73. No más de5 secciones en una envolvente.

## A.1.1 Error numbers generated by the compiler.

1. Number too large.
2. Semi-colon expected.
3. Undeclared identifier.
4. Identifier expected.
5. Use '=' not ':=' in a constant declaration.
6. '='expected.
7. This identifier cannot begin a statement.
8. ':=' expected.
9. ')' expected.
10. Wrong type.
11. '.' expected.

12. Factor expected.
13. Constant expected.
14. This identifier is not a constant.
15. 'THEN' expected.
16. 'DO' expected.
17. 'TO' or 'DOWNTO' expected.
18. '(' expected.
19. Cannot write this type of expression.
20. 'OF' expected.
21. ',' expected.
22. ':' expected.
23. 'PROGRAM' expected.
24. Variable expected since parameter is a variable parameter.
25. 'BEGIN' expected.
26. Variable expected incall to READ.
27. Cannot compare expressions of this type.
28. Should be either type INTEGER or type REAL.
29. Cannot read this type of variable.
30. This identifier is not a type.
31. Exponent expected inreal number.
32. Scalar expression (notnumeric) expected.
33. Null strings not allowed (useCHR(0)).
34. '[' expected.
35. ']' expected.
36. Array index type must be scalar.
37. '..' expected.
38. ']' or ',' expected in ARRAY declaration.
39. Lowerbound greater than upperbound.
40. Settoolarge (morethan256possible elements).
41. Function result must be type identifier.
42. ',' or ']' expected in set.
43. '..' or ',' or ']' expected in set.
44. Type of parameter must be a type identifier.
45. Null set cannot be the first factor in a non-assignment statement.
46. Scalar (including real) expected.
47. Scalar (not including real) expected.
48. Sets incompatible.
49. '<'and'>' cannot be used to compare sets.
50. 'FORWARD', 'LABEL', 'CONST', 'VAR', 'TYPE' or 'BEGIN'expected.
51. Hexadecimal digit expected.
52. Cannot POKE sets.
53. Array too large (>64K).
54. 'END' or ';' expected in RECORD definition.
55. Field identifier expected.
56. Variable expected after 'WITH'.
57. Variablein WITH must be of RECORD type.
58. Field identifier has not had asociated WITH statement.
59. Unsigned integer expected after 'LABEL'.
60. Unsigned integer expected after 'GOTO'.
61. This label is at the wrong level.
62. Undeclared label.
63. The parameter of SIZE should be a variable.
64. Can only use equalitytests for pointers.
67. The only write parameter for integers with two ':'sise:m:H.
68. Strings may not contain end of line characters.
69. The parameter of NEW, MARK or RELEASE should be a variable of pointer type.
70. The parameter ofADDR should be a variable.
71. This parameter must be a procedure.
72. This parameter must be a parameterless procedure.
73. No more than 5 sections in an envelope.

### A.1.2 Mensajes de Error en Tiempo de Ejecución

Cuando el **explotador** detecta un error en tiempo de ejecución, mostrará uno de los siguientes mensajes, seguidos de 'at PC=XXXX' siendo XXXX la dirección en memoria que estaba registrada **en el contador de programa** cuando se detectó el error. A menudo el origen del error será obvio; pero si no lo es, consulta el listado de la compilación para ver en el programa fuente dónde apareció el error durante la ejecución del programa objeto resultado de la compilación; usando XXXX como dirección de acceso al listado. En ocasiones, eso no da el resultado correcto.

```
1. Halt
2. Overflow
3. Out of RAM
4. / by zero                    also generated by DIV.
5. Index too low
6. Index too high
7. Maths Call Error
8. Number too large
9. Number expected
1Ø. Line too long
11. Exponent expected
```

Los errores en tiempo de ejecución provocan que se detenga la ejecución del programa.

Más o menos, vienen a decir los mensajes lo siguiente:

1. Detención
2. Rebase
3. Sin suficiente memoria de escritura-lectura.
4. División por cero.
5. Indice demasiado bajo
6. Indice demasiado alto.
7. Error en la cita a rutinas matemáticas.
8. Número demasiado grande.
9. Esperado un número.
10. Línea demasiado larga.
11. Esperado exponente.

# APENDICE 2
# PALABRAS RESERVADAS
# E IDENTIFICADORES PREDEFINIDOS

## A2.1 Palabras Reservadas

```
AND       ARRAY       BEGIN     CASE      CONST      DIV     DO
DOWNTO    ELSE        END       FORWARD   FUNCTION   GOTO    IF
IN        LABEL       MOD       NIL       NOT        OF      OR
PACKED    PROCEDURE   PROGRAM   RECORD    REPEAT     SET     THEN
TO        TYPE        UNTIL     VAR       WHILE      WITH
```

## A2.2 Símbolos Especiales

Los siguientes símbolos se usan el Hisoft Pascal 4, y tienen un significado **reservado:**

```
+       -       *       /
=       < >     <       <=          >=      >
(       )       [       ]
{       }       (*      *)
^       :=      .       ,           ;       :
:
.       ..
```

## A2.3 Identificadores Predefinidos

Las siguientes **entidades** pueden considerarse como declaradas en un **bloque** que rodea a todo el programa, y que por tanto están disponibles a través del programa, a no ser que sean re-definidas por el programador dentro de un bloque interno al programa.

Para más información, véase la Sección 2.

```
CONST       MAXINT=32767;
TYPE        BOOLEAN=(FALSE,TRUE);
            CHAR {El repertorio ampliado de caracteres
            ASCII;
            INTEGER=-MAXINT..MAXINT;
            REAL {Un subconjunto de los números reales.
            Véase Sección 1.3.
VAR         ERRFLG, ERRCHK: BOOLEAN: RA, RB,
            RC, RD, RE, RF, RH, RL: CHAR;
            RAF, RBC, RDE, RHL: INTEGER;
```

| | |
|---|---|
| PROCEDURE | WRITE; WRITELN; READ; READLN; PAGE;<br>HALT; USER; POKE; INLINE;<br>OUT; NEW; MARK; RELEASE; TIN; TOUT;<br>AFTER; EVERY; SOUND: ONSQ;<br>EXTERNAL; ENV; ENT; |
| FUNCTION | ABS; SQR; ODD; RANDOM; ORD; SU[C;<br>PRED; INCH; EOLN;<br>PEEK; CHR; SQRT; ENTIER; ROUND;<br>TRUNC: FRAC; SIN; COS;<br>TAN; ARCTAN; EXP; LN; ADDR; SIZE;<br>INP; REMAIN; INITEVENT; |

# APENDICE 3
# REPRESENTACION DE DATOS Y ALMACENAJE

## A3.1 Representación de Datos

Los siguientes comentarios detallan cómo se representan **internamente** los datos por el programa Hisoft Pascal.

La información de la cantidad de almacenamiento requerida en cada caso, debiera ser útil para la mayoría de los programadores la función de **tamaño** (SIZE puede usarse según la Sección 2.3.6.7); aquéllos que intenten mezclar programas en Pascal y en código máquina pueden necesitar detalles adicionales.

### A3.1.1 Enteros

Los enteros -Integers- ocupan 2 bytes de memoria cada uno, en notación de **complemento a doses.** Ejemplos:

```
    1   ≡  #ØØØ1
  256   ≡  #Ø1ØØ
 -256   ≡  #FFØØ
```

Los registros standard del Z80 usados por el compilador para guardar enteros son los HL.

### A3.1.2 Caracteres, Datos Booleanos y otros Scalares

Todos ellos ocupan 1 byte en memoria cada uno, en notación **binaria sin-signo,** pura.

Para los caracteres se usan los 8 bits del ASCII ampliado.

```
'E'   ≡   #45
'['   ≡   #5B
```

Para los datos Booleanos, dado que:

```
ORD(TRUE)=1      por tanto TRUE se representa por 1
ORD(FALSE)=Ø     por tanto FALSE se representa por 0.
```

El registro standard del Z80 usado por el compilador para estos datos es el A.

### A3.1.3 Reales

La notación **mantisa, exponente** se usa de manera similar a la empleada en la notación científica standard -sólo que usando el sistema binario en lugar del **denario** (base 10). Ejemplos:

$2 \equiv 2*10^0$ ó $1.0_2*2^1$

$1 \equiv 1*10^0$ ó $1.0_2*2^0$

$-12.5 \equiv -1.25*10^{1}$ ó $-25*2^{-1}$

$\equiv -11001_2*2^{-1}$

$-1.1001_2*2^{3}$ cuando normalizado.

$0.1 \equiv 1.0*10^{-1}$ ó $\frac{1}{10} \equiv \frac{1}{1010_2} \equiv \frac{0.1_2}{1010_2}$

así que ahora necesitamos hacer una división binaria...

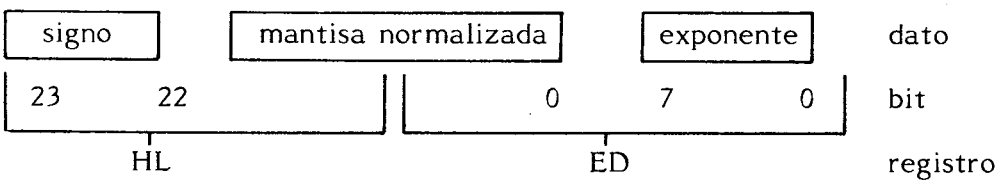
```
        Ø.ØØØ11ØØ
1Ø1 | Ø.1ØØØØØØØØØØØØØØ
        1Ø1
        11Ø
        1Ø1
           1ØØØ
            1Ø1
```

en este momento veremos que la fracción es periódica

$= \frac{0.1_2}{1010_2} = 0.0001\dot{1}\dot{0}0_2$

$1.1001\dot{1}\dot{0}0*2^{-4}$ respuesta

Así que ¿cómo usamos los resultados anteriores para representar estos números en memoria? Bien, primeramente reservamos **4 bytes** de almacenaje para cada dato **real** según el formato siguiente:




signo: el signo de la mantisa: 1=negativo, 0=positivo.

mantisa normalizada: la mantisa normalizada en la forma 1.xxxxxx... con el bit superior (bit 22) siempre 1, excepto cuando representa el **cero** (HL=0, DE=0).

exponente: el exponente en forma binaria de complemento a doses

Por lo tanto:

```
2  ≡      Ø 1ØØØØØØ ØØØØØØØØ ØØØØØØØØ ØØØØØØØ1   (#4Ø,#ØØ,#ØØ,#Ø1)
1  ≡      Ø 1ØØØØØØ ØØØØØØØØ ØØØØØØØØ ØØØØØØØØ   (#4Ø,#ØØ,#ØØ,#ØØ)
-12.5 ≡   1 11ØØ1ØØ ØØØØØØØØ ØØØØØØØØ ØØØØØØ11   (#E4,#ØØ,#ØØ,#Ø3)
Ø.1 ≡     Ø 11ØØ11Ø Ø11ØØ11Ø Ø11ØØ11Ø 11111ØØ   (#66,#66,#66,#FC)
```

Y así, recordando que HL y DE se usan para cargar los números reales, tendríamos que cargar los registros de la siguiente manera:

```
2        ≡      LD HL,#4ØØØ
                LD DE,#Ø1ØØ

1        ≡      LD HL,#4ØØØ
                LD DE,#ØØØØ

-12.5    ≡      LD HL,#E4ØØ
                LD DE,#Ø3ØØ

Ø.1      ≡      LD HL,#6666
                LD DE,#FC66
```

Observa bien: los reales se depositan en memoria en el orden ED LH.

### A3.1.4 Fichas y arrays

Las **fichas** (record) usan la misma cantidad de memoria como el total de sus campos componentes.

Las **arrays:** si **n** = número de elementos y
**s** = tamaño de cada elemento

el número de bytes ocupado en memoria sería de **n*s**

e.g. un ARRAY[1..1Ø] OF INTEGER requiere 1Ø*2=2Ø bytes; un ARRAY[2..12,1..1Ø] OF CHAR tiene 11*1Ø=11Ø elementos y por tanto requiere 110 bytes.

### A3.1.5 Grupos

Los grupos (sets) se depositan como **series de bits** y por tanto si el tipo base del grupo tiene n elementos, el número de bytes usado es (n-1) DIV 8+1. Ejemplos:

un SET OF CHAR requiere (256-1) DIV 8 + 1 = 32 bytes; un SET OF (azul, verde, amarillo) requiere (3-1) DIV 8 + 1 = 1 byte... (aunque en realidad sólo emplea 3 bits de dicho byte).

### A3.1.6 Punteros

Los datos de tipo **puntero** ocupan 2 bytes ya que contienen la **dirección** (en formato Intel, i.e. primero el byte bajo) de la variable hacia la que señalan.

## A3.2 Almacenamiento de Variables en Tiempo de Ejecución

Hay 3 casos en que el usuario necesita la información sobre cómo están depositados los valores de las variables, cuando está el programa en **explotación.**

a. Variables globales - declaradas en el bloque principal del programa
b. Variables locales - declaradas en un bloque interno
c. Parámetro y resultados devueltos - traspasados hacia o desde los procedimientos y las funciones.

Estos casos individuales se comentan a continuación, y un ejemplo de cómo puede usarse esta información se da en el Apéndice 4.

Variables globales

Una **variable global** tiene su espacio adjudicado a partir de la cima del espacio ocupado como **percha** en tiempo de ejecución, y progresando en sentido descendente. E.g. si dicho depósito temporal comienza en la dirección #B000 y las variables del programa principal son:

```
VAR       i  : INTEGER;
          ch : CHAR;
          x  : REAL;
```

entonces se **apilarán** ordenadamente de manera que:

**i** (que ocupa 2 bytes por ser **entero**) estará depositada en las direcciones #B00-2 y #B000-1, i.e. en #AFFE y #AFFF.

**ch** (que ocupa 1 byte por ser un **carácter**) estará depositada en la dirección #AFFE-1, i.e. en #AFFD.

**x** (que ocupa 4 bytes por ser **real**) estará colocada en #AFF9, #AFFA, #AFFB y #AFFC.

Variables locales

Las variables locales **no son accesibles** a través del espacio usado como **percha** tan fácilmente. En lugar de eso, el registro IX se carga con la dirección de comienzo de cada **bloque interno** de manera que **IX-4** señale hacia el comienzo de las variables locales del bloque. E.g.:

```
PROCEDURE       test;
VAR             i,j:INTEGER;
```

y por lo tanto:

**i** (que ocupa 2 bytes por ser entero) estará depositado en IX-4-2 e IX-4-1, i.e. IX-6 e IX-5; y la variable **j** estará situada en IX-8 e IX-7.

Parámetros y resultados devueltos

Los parámetros **constantes** ("de valor") se tratan igual que las variables locales, y como ellas, cuanto más pronto se haya declarado el parámetro, mayor será la dirección que tenga en memoria. Sin embargo, a diferencia de las variables, la dirección **baja** (no la alta) es la que está prefijada, y lo está como (IX+2). E.g.

```
PROCEDURE test(i:REAL; j:INTEGER);
```

Por lo tanto:

**j** (depositada en primer lugar) estará en IX+2 e IX+3
**i** estará en IX+4, IX+5, IX+6, e IX+7

Los parámetros **variables** se tratan justamente igual que los parámetros constantes, exceptuando que siempre se les adjudican 2 bytes, y que dichos 2 bytes contienen la dirección de la variable. E.g.:

```
PROCEDURE test  (i:INTEGER; VARx:REAL);
```

por tanto:

la **referencia** a la variable **x** se coloca en IX+2 e IX+3; estas direcciones contienen la dirección **actual** donde está depositado el valor de **x.** El valor de **i** estará en IX+4 e IX+5.

Los **valores devueltos** -los resultados- de las funciones se colocan por encima del primer parámetro que haya en memoria, e.g.:

```
FUNCTION test (i:INTEGER):REAL;
```

hará que **i** esté en IX+2 e IX+3 y que se reserve espacio para el valor devuelto por la función en IX+4, IX+5, IX+6 e IX+7.

# APENDICE 4
# ALGUNOS EJEMPLOS DE PROGRAMAS EN HISOFT PASCAL

Los siguientes programas deben ser estudiados cuidadosamente, si tienes alguna duda en cómo programar usando el Hisoft Pascal.

```
PROGRAM FACTOR

1Ø    (*Programa para mostrar el uso de recursión*)
2Ø
3Ø    PROGRAMFACTOR;
4Ø
5Ø    (*Este programa calcula el factorial de un número
       impuesto desde el teclado
6Ø    1) usando un método recursivo y 2) usando un
       método reiterativo.*)
7Ø
8Ø    TYPE
9Ø    POSINT=0..MAXINT;
1ØØ
11Ø   VAR
12Ø   METHOD:CHAR;
13Ø   NUMBER:POSINT;
14Ø
15Ø   (*Algoritmo recursivo.*)
16Ø
17Ø   FUNCTIONRFAC(N:POSINT):INTEGER;
18Ø
19Ø   VARF:POSINT;
2ØØ
21Ø   BEGIN
22Ø   IF N>1 THEN F:= N * RFAC(N-1) (*RFAC se cita N
       veces*)
23Ø   ELSE F :=1;
24Ø   RFAC:=F
25Ø   END;
26Ø
27Ø   (*Solución reiterativa*)
28Ø
29Ø   FUNCTIONIFAC(N:POSINT):INTEGER;
3ØØ
31Ø   VARI,F:POSINT;
32Ø   BEGIN
33Ø    F:=1;
34Ø    FORI:=2TONDOF:=F*1;(*Bucle simple*)
35Ø    IFAC:=F
36Ø   END;
37Ø
```

```
38Ø   BEGIN
39Ø   REPEAT
4ØØ   WRITE('Dime método (I o R) y número');
41Ø   READLN;
42Ø   READ(METHOD,NUMBER);
43Ø   IF METHOD = 'R'
44Ø   THEN WRITELN(NUMBER,'! = ',RFAC(NUMBER))
45Ø   ELSE WRITELN(NUMBER,'! = ',IFAC(NUMBER))
46Ø   UNTIL NUMBER=Ø
47Ø   END.
```

```
PROGRAMREV

1Ø    {Programa para listar líneas de un fichero en orden
       inverso.
2Ø    Muestra el uso de punteros, fichas, MARK y RELEASE.}
3Ø
4Ø    PROGRAM ReverseLine;
5Ø
6Ø    TYPEelem=RECORD          {Crea estructura de lista-
                                encadenada}
7Ø    next:↑elem;
8Ø    ch:CHAR
9Ø    END;
1ØØ   link=↑elem;
11Ø
12Ø   VARprev,cur,heap:link; {todos son punteros hacia
       'elem'}
13Ø
14Ø   BEGIN
15Ø   REPEAT                          {haga esto muchas veces}
16Ø   MARK(heap)              {asigne cima de cúmulo a 'heap'}
17Ø   prev:=NIL;     {todavía no alude a ninguna variable}
18Ø   WHILE NOT EOLN DO
19Ø   BEGIN
2ØØ   NEW(cur);      {reserva espacio para un nuevo 'elem'}
21Ø   READ(cur↑.ch);        {e impone en el campo de dicho}
22Ø                {'elem' el carácter recogido del teclado}
23Ø   cur↑.next:=prev;       {carga el puntero en la ficha}
24Ø   prev:=cur    {preserva el puntero a la ficha previa}
25Ø   END;
26Ø
27Ø   {Escribe la línea en sentido inverso explorando
28Ø   la lista de fichas que ha construido.}
29Ø
3ØØ   cur:=prev;
31Ø   WHILE cur<>NILDO                    {NIL es el primero}
32Ø   BEGIN
33Ø   WRITE(cur↑.ch);{Escribe este campo i.e. el carácter}
34Ø   cur:=cur↑.next          {Señala hacia el campo previo}
35Ø   END;
36Ø   WRITELN;
37Ø   RELEASE(heap);     {Libera el espacio de la variable
                                                   dinámica}
38Ø   READLN                      {Espera otra línea de texto}
39Ø   UNTIL FALSE                       {Usa [ESC] para salir}
4ØØ   END.
```

PROGRAM TINTOUT

Programa para ilustrar el uso de TIN y TOUT para transferencia de datos a cinta. El programa construye un listín telefónico muy simple en la cinta y luego lo vuelve a recuperar de ella. Deberás escribir cualquier procedimiento de búsqueda que se requiera.

```
PROGRAM TINTOUT;

CONST
  Size=1Ø;
TYPE
  Entry=RECORD
     Name:ARRAY[1..1Ø]OF CHAR;
     Number:ARRAY[1..1Ø]OF CHAR
   END;
VAR
  Directory:ARRAY[1..Size]OF Entry;
  I:INTEGER;
BEGIN
(Establece el directorio..
FORI:= 1 TO Size DO
BEGIN
  WITHDirectory[I] DO
  BEGIN
    WRITE('Dime nombre');
    READLN;
    READ(Name);
    WRITELN;
    WRITE('Dime número');
    READLN;
    READ(Number);
    WRITELN
  END
END;
{Para depositar el directorio en cinta usa..}
TOUT('Directory',ADDR(Directory),SIZE(Directory))
{Ahora para recuperar el listín haz lo siguiente..}
TIN('Directory',ADDR(Directory))
{Y ahora puedes procesar el directorio como tú
desees....}
END.
```

```
PROGRAM DIRTY.

1Ø    {Programa para mostrar cómo 'ensuciarte las manos'!
2Ø    i.e. cómo modificar variables en Pascal usando
       código máquina.
3Ø    Demuestra PEEK, POKE, ADDR e INLINE}
4Ø
5Ø    PROGRAM divmult2;
6Ø
7Ø    VAR r:REAL;
8Ø
9Ø    FUNCTION divby2(x:REAL):REAL;          {función para
                                            dividir por 2..
1ØØ                                          ..rápidamente}
11Ø   VARi:INTEGER;
12Ø   BEGIN
13Ø   i:=ADDR(x)+1;       {Apunta hacia el exponente de x}
14Ø   POKE(i,PRED(PEEK(i,CHAR)));            {Decrementa el
                                            exponente de x}
15Ø   {véase Apéndice 3.1.3}
16Ø   divby2:=x
17Ø   END;
18Ø
19Ø   FUNCTION multby2(x:REAL):REAL;         {función para
                                         multiplicar por 2..
2ØØ                                           ..rápidamente}
21Ø   BEGIN
22Ø   INLINE(#DD,#34,3);    {INC (IX+3)- el exponente de x
23Ø                                  - véase Apéndice 3.2}
24Ø   multby2:=x
25Ø   END;
26Ø
27Ø   BEGIN
28Ø   REPEAT
29Ø   WRITE('Teclea el número r ');
3ØØ   READ(r);              {No se necesita READLN - véase
31Ø                                      Sección 2.3.1.4}
32Ø
33Ø   WRITELN('r dividido por dos es',divby2(r):7:2);
34Ø   WRITELN('r multiplicado por dos es',multby2(r):7:2)
35Ø   UNTIL r=Ø
36Ø   END.
```

# APENDICE 5
# GRAFICOS DE TORTUGA EN HISOFT PASCAL

El **paquete** para Gráficos de Tortuga está contenido en la otra cara de tu cinta maestra en cassette bajo el nombre TURTLE.

El paquete está escrito en Pascal y puede ser cargado dentro del editor Hisoft Pascal usando el comando 'G,,TURTLE'. Con eso se cargarán el **segmento** con el programa para gráficos de tortuga y lo añadirá a cualquier programa existente: observa que con el propósito de que funcione correctamente, los Gráficos de Tortuga deben ir precedidos por una cabecera normal PROGRAM y una declaración VAR -las declaraciones TYPE, CONST y LABEL son opcionales y no debe haber ningún Procedimiento ni Función declarada previamente a la inclusión del paquete para Gráficos de Tortuga.

El paquete TURTLE se suministra conteniendo un programa de demostración y para que funcione simplemente teclea:

g,,TURTLE[**ENTER**]
**C**[**ENTER**]

y responde **y** (yes=sí) a la cuestión Run? al final de la compilación.

Para extraer el **núcleo** de las rutinas para Gráficos de Tortuga, que se documentan a continuación, debes:

d1Ø,4Ø[**ENTER**]
d115Ø,232Ø [**ENTER**]
p1,114Ø,turtle [**ENTER**]

aunque desde luego, puede que quieras conservar alguno de los otros procedimientos y funciones que son parte del programa de demostración; están colocados entre los números de línea 1150 a 2320.

Como en la mayoría de las implementaciones de los Gráficos de Tortuga, el TURTLE del Hisoft Pascal crea una criatura imaginaria en la pantalla que el usuario puede mover por cualquier lado vía algunos comandos muy simples. Esta **tortuga** puede ir dejando una **estela** (en diversos colores) o puede hacerse invisible). El **rumbo** y la **posición** de la tortuga se conservan como variables globales que se actualizan cuando se mueve o se gira dicha criatura; obviamente estas variables pueden examinarse o alterarse en cualquier momento.

Las facilidades disponibles son las siguientes:

**Variables Globales**

heading

se usa para conservar el **rumbo**, i.e. el valor angular de la dirección en que la tortuga está orientada en este momento. Toma cualquier valor REAL en grados, y puede inicializarse a cero con el procedimiento TURTLE (véase más adelante).

El valor 0 corresponde a la orientación hacia el **Este,** de manera que después de recurrir al procedimiento TURTLE, la tortuga está orientada de izquierda a derecha. A medida que el rumbo crece a partir de cero, la tortuga gira en sentido contrario a las agujas del reloj.

Xcor, Ycor

son las coordenadas actuales **(x,y)** que definen la posición de la tortuga en la pantalla. La pantalla de gráficos del CPC464 tiene un tamaño de 640*200 puntos y la tortuga puede estar situada en cualquier punto dentro de ese área, suponiendo que estás trabajando en el modo de mayor resolución; cuando usas modos de resolución menor, todavía puedes especificar 640*400 pero la resolución no será de un punto.

Inicialmente **X**cor e **Y**cor están sin valor definido; usa el procedimiento TURTLE para darles el valor inicial a 300 y 200 respectivamente, así colocarás a la tortuga en el medio de su "terreno de juego".

penstatus

una variable Booleana que refleja el **estado de la pluma** en cada momento (i.e. las condiciones de la **estela** dejada por la tortuga). TRUE(=cierto) significa que la pluma está **baja** (deja rastro); FALSE(=falso) significa que la pluma está subida (no deja rastro).

**Procedimientos**

Los procedimientos disponibles son:

INK(I,C1,C2:INTEGER)

que fija la tinta **I** para que tenga los valores de color especificados por C1 y C2. Si C1 = C2 entonces la tinta será de un color **estacionario.** En los demás casos, alternará entre dos colores.

```
INK(1,12,12);      fija la tinta 1 a un amarillo estacionario.
INK(Ø,16,21);      fija la tinta 0 a un parpadeo entre rosa y lima!
```

PAPER(I:INTEGER)

fija el color del **fondo** (el papel) de la pantalla para que corresponda al color (o los colores) asociados con la **tinta I,** que ha de ser un entero.

PEN(I:INTEGER)

fija el color de la **pluma** (el frente) al color o colores asociados con la tinta **I.**

PENDOWN(I:INTEGER)

fija el estado de la tortuga de manera que dejará una **estela** en el color asociado con el parámetro **I** de tinta.

Este procedimiento asigna TRUE a la variable global penstatus.

PENUP

después de citar este procedimiento, los movimientos posteriores de la tortuga no dejarán ningún rastro en pantalla. Util para desplazarla desde una sección del gráfico hasta otra.

PENUP asigna el valor FALSE(=falso) a la variable global penstatus.

SETHD(A:REAL)

toma un parámetro REAL que es asignado como valor de la variable global **heading,** fijando así el **rumbo,** o dirección que seguirá la tortuga en su movimiento. Recuerda que un rumbo de cero corresponde al EAST, 9Ø al NORTH, 18Ø al WEST y 27Ø al SOUTH (que obviamente corresponden a Este, Norte, Oeste y Sur).

SETXY(X,Y:REAL)

fija como posición **absoluta** de la tortuga dentro del área de gráficos, la correspondiente a las coordenadas **(x,y)**. No se efectúa ninguna comprobación dentro de este procedimiento para averiguar si **(X,Y)** está fuera de límites; es el programa grabado en la ROM del sistema el que lo comprueba.

FWD(L:REAL)

**avanza** (forward) la tortuga **L** unidades en la dirección indicada por el rumbo corriente. Las unidades de **longitud** corresponden a un elemento de imagen gráfico, redondeado hacia arriba o hacia abajo según sea necesario.

BACK(L:REAL)

mueve la tortuga **L** unidades en la dirección opuesta al rumbo actual (backward=retrocede). El rumbo no se ve alterado, aunque el movimiento es casi similar a si se hubiera producido un giro de -180.

TURN(A:REAL)

**gira** un ángulo, i.e. cambia el rumbo de la tortuga en **A** grados, sin moverla del sitio que ocupa. El rumbo se incrementa al girar en sentido contrario a las agujas del reloj.

MODE(M:INTEGER)

establece para la pantalla el modo **M,** siendo **M** un entero con un valor entre 0 y 2, que corresponde a los modos de pantalla disponibles en el BASIC.

RIGHT(A:REAL)

es una alternativa a un **giro a derecha,** y cambia el rumbo que seguirá la tortuga, girando en el sentido de las agujas del reloj, un ángulo de **A** grados.

LEFT(A:REAL)

este comando es idéntico al de giro TURN, y se ha provisto simplemente para mayor conveniencia y compatibilidad con la orden contraria RIGHT.

ARCR(R:REAL,A:INTEGER)

la tortuga se mueve a través de un **arco circular** cuyo tamaño viene estipulado por el **radio R.** La longitud del arco está determinada por **A,** que representa el **ángulo** girado entre los extremos del arco (el subtendido desde el centro del círculo) en el sentido de las agujas del reloj. Típicamente **R** puede ser 0.5.

TURTLE

este procedimiento simplemente fija el estado inicial de la **tortuga;** la sitúa en el medio de la pantalla y orientada al **Este** (rumbo de 0), sobre un fondo amarillo brillante y dejando una estela azul. Recuerda que el estado de la tortuga no está definido en un principio, de manera que este procedimiento se usa a menudo al comienzo de un programa.

Aquí concluye la lista de facilidades disponibles con TURTLE del Hisoft Pascal; aunque simple en implementación y utilización, hallarás que los Gráficos de Tortuga son capaces de producir dibujos muy complejos con una elevada rapidez. Para abrirte el apetito, te presentamos ahora algunos programas de ejemplo. Recuerda que debes tener Hisoft Pascal implantado en memoria, antes de introducir los programas.

**Ejemplo de Programas**

En todos los ejemplos de programas dados aquí, suponemos que ya has cargado en memoria el Hisoft Pascal y usado el comando 'G,,TURTLE' para traer de la cinta el **paquete** para Gráficos de Tortuga que comienza en la línea 10 y finaliza en la línea 1140, y que has suprimido las partes del paquete que forman el programa de demostración, tal como te hemos detallado. Ahora puedes proceder con los ejemplos:

1. CIRCUNFERENCIAS

```
   1 PROGRAM CIRCLES;
   2 VAR I:INTEGER:

115Ø BEGIN
116Ø TURTLE;
117Ø FOR I:=1 TO 9 DO
118Ø BEGIN
119Ø ARCR(Ø.5,36Ø);
12ØØ RIGHT(4Ø)
121Ø END
122Ø END.
```

2. ESPIRALES

```
   1 PROGRAM SPIRALS;
   2 VAR

115Ø PROCEDURE SPIRALS ( L,A:REAL );
116Ø BEGIN
117Ø FWD(L);
118Ø RIGHT(A);
119Ø SPIRALS(L+1,A)
12ØØ END;
121Ø BEGIN
1220 TURTLE;
123Ø SPIRALS(9,95)  (*o (9,9Ø) o (9,121)...*)
124Ø END.
```

3. FLORES

```
   1 PROGRAM FLOWER;
   2 VAR

115Ø PROCEDURE PETAL (S:REAL);
116Ø BEGIN
117Ø ARCR(S,6Ø);
118Ø LEFT(12Ø);
119Ø ARCR(S,6Ø)
12ØØ LEFT(12Ø)
121Ø END;
122Ø PROCEDURE FLOWER (S:REAL);
123Ø VAR I:INTEGER;
124Ø BEGIN
125Ø FOR I =1 TO 6 DO
126Ø BEGIN
127Ø PETAL(S);
128Ø RIGHT(6Ø)
129Ø END
13ØØ END;
131Ø BEGIN TURTLE;
132Ø SETXY(127,6Ø)
133Ø LEFT(9Ø); FWD(1Ø);
134Ø RIGHT(6Ø); PETAL(Ø.2);
135Ø LEFT(6Ø); PETAL(Ø.2);
136Ø SETHD(9Ø); FWD(4Ø);
137Ø FLOWER(Ø.4)
138Ø END.
```

Para avanzar y ampliar el estudio de los Gráficos de Tortuga te recomendamos el excelente (aunque caro) libro "Turtle Geometry" por Harold Abelson y Andrea di Sessa, publicado por MIT Press, ISBN 0-262-01063-1.

# APENDICE 6
# RUTINAS UTILES GRABADAS EN EL CPC464

A continuación, se da un listado de un conjunto de procedimientos y funciones en Pascal que te permiten **recurrir,** desde dentro del Hisoft Pascal, a las diversas rutinas grabadas en la ROM del CPC464. Debieras elegir y utilizar sólo las rutinas que requieres para cualquier aplicación. Las rutinas están diseñadas para ser **auto-documentadas.**

## Documentación sobre procedimientos de Firmware

```
1@ (* getjoy es como la FUNCION JOY del BASIC; su parametro
2@    debera ser un @ o un 1; entrega un valor 'calibrado en binario'
3@    al igual que en el BASIC *)
4@
5@ FUNCTION getjoy(stick:integer):integer;
6@ BEGIN
7@  user(#bb24);
8@  IF stick=1 THEN ra:=r1;
9@  getjoy:=ord(ra)
1@@ END
11@
12@  (* txtinitialise restaura los valores iniciales de la pantalla de texto:
13@     El papel de texto se fija a tinta @.
14@     La pluma de texto se fija a tinta 1.
15@     La pantalla de texto se fija a toda la pantalla.
16@     El cursor de texto esta facultado pero oculto.
17@     El modo de escritura de caracteres se fija a opacos.
18@     Se habilita la VDU.
19@     El modo de escritura de caracteres graficos se cancela.
2@@     El cursor se mueve hasta la esquina inferior izquierda de pantalla *)
21@
22@ PROCEDURE txtinitialise;
23@ BEGIN
24@  user(#bb4e)
25@ END;
26@
27@ (* txtout saca un caracter o un codigo de control hasta
28@    la VDU de texto, SIN que el Pascal trate los codigos de control.
29@    Eso debera usarse cuando, por ejemplo se establecen los colores
3@@    de tinta o papel usando codigos de control. *)
31@
32@ PROCEDURE txtout(c:char);
33@ BEGIN
34@  ra:=c;
35@  user(#bba)
36@ END;
37@
38@ (* txtrdchar lee el caracter que en la pantalla esta situado en la
39@    posicion corriente del cursor. *)
4@@
41@ FUNCTION txtrdhar:char;
42@ BEGIN
43@  user(#bb6@);
44@  txtrdchar:=ra
45@ END;
46@
47@ (* winenable faculta y fija una ventana de texto en pantalla *)
48@
49@ PROCEDURE winenable(col1,col2,row1,row2:integer);
5@@ BEGIN
```

```
51@  rh:=chr(col1); rd:=chr(col2);
52@  rl:=chr(row1); re:=chr8row2);
53@  user(#bb66)
54@ END;
55@
56@ (* getwindow devuelve el tamano de la ventana corriente,con sus
57@    parametros variables de columnas y filas *)
58@
59@ PROCEDURE getwindow(VAR col1,col2,row1,row2:integer);
6@@ BEGIN
61@  user(#bb69);
62@  col1:=ord(rh); col2:=ord(rd);
63@  row1:=ord(rl); row2:=ord(re)
64@ END;
65@
66@ (* clearwindow limpia la ventana de texto corriente*)
67@
68@ PROCEDURE clearwindow;
69@ BEGIN
7@@  use(#bb6c)
71@ END;
72@
73@ (* setcolumn fija la posicion horizontal del cursor*)
74@
75@ PROCEDURE setcolumn(c:integer);
76@ BEGIN
77@  ra:=chr(c);
78@  user(#bb6f)
79@ END;
8@@
81@ (* setrow fija la posicion vertical del cursor *)
82@
83@ PROCEDURE setrow(r:integer);
84@ BEGIN
85@  ra:=chr(r);
86@  user(#bb72)
87@ END;
88@
89@ (* setcursor fija la posicion del cursor en la dada por la
9@@    columna y fila especificada en el procedimiento *)
91@
92@ PROCEDURE setcursor(c,r:integer);
93@ BEGIN
94@  rh:=chr(c); rl:=chr(r);
95@  user(#bb75)
96@ END;
97@
98@ (* getcursor entrega la posicion corriente del cursor, en columna y
99@    fila, y la cuenta de 'corrimiento'. Esta cuenta no tiene ningun
1@@@   significado absoluto pero se incrementa si la ventana se 'corre
```

```
1010   hacia abajo' y se decrementa si se 'corre hacia arriba. *)
1020
1030 PROCEDURE getcursor(VAR col,row,roll:integer);
1040 BEGIN
1050  user(#bb78);
1060  col:ord(rh); row:=ord(rl); roll:=(ra)
1070 END;
1080
1090 (* curenable faculta el cursor.El cursor se
1100    muestra en pantalla solo si esta facultado y puesto. *)
1110
1120 PROCEDURE curenable;
1130 BEGIN
1140  user(#bb7b);
1150 END;
1160
1170 (* curdisable cancela el cursor. *)
1180
1190 PROCEDURE curdisable;
1200 BEGIN
1210  user(#bb7e)
1220 END;
1230
1240 (* curon pone el cursor y lo muestra, solo si esta facultado. *)
1250
1260 PROCEDURE curon;
1270 BEGIN
1280  user(#bb81)
1290 END;
1300
1310 (* curoff quita el cursor. *)
1320
1330 PROCEDURE curoff;
1340 BEGIN
1350  user(#bb84)
1360 END;
1370
1380
1390 (* txtsetpen fija la pluma de texto para escribir caracteres. *)
1400
1410 PROCEDURE txtsetpen(INK:integer);
1420 BEGIN
1430  ra:=chr(INK);
1440  user(#bb90)
1450 END;
1460
1470 (* txtgetpen entrega el valor actual de la pluma de texto. *)
1480
1490 FUNCTION txtgetpen:integer;
1500 BEGIN
```

```
151@  user(#bb93);
152@  txtgetpen:=ord(ra)
153@ END;
154@
155@ (* txtsetpaper fija la tinta a usar con el papel, fondo. *)
156@
157@ PROCEDURE txtsetpaper(INK:integer);
158@ BEGIN
159@  ra:=chr(INK);
160@  user(#bb96)
161@ END;
162@
163@ (* gettxtpaper entrega el valor actual del papel de texto. *)
164@
165@ FUNCTION gettxtpaper:integer;
166@ BEGIN
167@  user(#bb99);
168@  gettxtpaper:=ord(ra)
169@ END;
170@
171@ (* txtinverse intercambia las tintas de pluma y papel. *)
172@
173@ PROCEDURE txtinverse;
174@ BEGIN
175@  user(#bb9c)
176@ END;
177@
178@ (* txtsetback hace que se use el modo transparente si este
179@    parametro es cierto; i.e. no escribe el fondo. Si este
180@    parametro es falso, usa el modo opaco en que
181@    se usa el color de fondo. *)
182@
183@ PROCEDURE txtsetback(b:boolean);
184@ BEGIN
185@  ra:=chr(ord(b));
186@  user(#bb9f)
187@ END;
188@
189@ (* txtgetback entrega el valor cierto si se esta usando
190@    el modo transparente; falso si se esta usando el modo opaco. *)
191@
192@ FUNCTION txtgetback:boolean;
193@ BEGIN
194@  user(#bba2);
195@  txtgetback:= ra = chr(1)
196@ END;
197@
198@ (* txtgetmatrix entrega la direccion de la matriz que da la forma
199@    al caracter mencionado como parametro. *)
```

```
2000
2010 FUNCTION txtgetmatrix(c:char):integer;
2020 BEGIN
2030  ra:=c;
2040  user(#bba5);
2050  txtgetmatrix:=rhl
2060 END;
2070
2080 (* txtsetmatrix copia la matriz que da forma al caracter, en la
2090    direccion correspondiente al caracter que se da como parametro. *)
2100
2110 PROCEDURE txtsetmatrix(c:char;adr:integer);
2120 BEGIN
2130  ra:=c;
2140  rhl:=adr;
2150  user(#bba8)
2160 END;
2170
2180 (* setmtable fija la direccion de la tabla de matrices que dan
2190    forma a los caracteres definidos por el usuario, a una nueva direccion.
2200    Su parametro c se usa como el numero que corresponde al caracter
2210    mas bajo a usar en la tabla. Si este parametro no esta en la banda
2220    0 a 255, la tabla de caracteres definida por el usuario se considera
2230    como vacia. Normalmente se usan arrays para almacenar tales matrices
2240    y se usa la FUNCION addr para traspasar las direcciones
2250    de la tabla. *)
2260
2270 PROCEDURE setmtable(c,adr:integer);
2280 BEGIN
2290  rde:=c; rhl:=adr;
2300  user(#bbab)
2310 END;
2320
2330 (* txtstrselect elige el numero de cauce vigente, que estara
2340    en la bamda 0..7. Muchos atributos de la VDU
2350    de texto pueden fijarse por separado para diversos cauces. *)
2360
2370 PROCEDURE txtstrselect(s:integer);
2380 BEGIN
2390  ra:=chr(s);
2400  user(#bbb4)
2410 END;
2420
2430 (* txtswapstreams canjea los atributos de dos cauces. *)
2440
2450 PROCEDURE txtswapstreams(s1,s2:integer);
2460 BEGIN
2470  rb:=chr(s1); rc:=chr(s2);
2480  user(#bbb7)
```

```
249@ END;
25@@
251@ (* grainitialise establece valores iniciales de la pantalla de graficos
252@    Fija el papel de graficos a tinta @
253@    Fija la pluma de graficos a tinta 1
254@    Fija el origen en la esquina inferior izquierda
255@    Mueve la posicion corriente al origen de la ventana
256@    Fija la ventana de graficos para que cubra toda la pantalla
257@    La ventana de graficos no se limpia. *)
258@
259@ PROCEDURE grainitialise;
26@@ BEGIN
261@  user(#bbba)
262@ END;
263@
264@ (* gramoveabsolute mueve la posicion corriente hasta la posicion dada
265@    en coordenadas absolutas. *)
266@
267@ PROCEDURE gramoveabsolute(x,y:integer);
268@ BEGIN
269@  rde:=x; rhl:=y;
27@@  user(#bbc@)
271@ END;
272@
273@ (* gramoverelative mueve la posicion corriente hasta una posicion dada
274@    por sus coordenadas relativas. *)
275@
276@ PROCEDURE gramoverelative(x,y:integer);
277@ BEGIN
278@  rde:=x; rhl:=y;
279@  user(#bbc3);
28@@ END;
281@
282@ (* graaskcursor 'pregunta' la posicion corriente del cursor, y da los
283@    resultados en los parametros variables. *)
284@
285@ PROCEDURE graaskcursor(VAR x,y:integer);
286@ BEGIN
287@  user(#bbc6);
288@  x:=rde; y:=rhl
289@ END;
29@@
291@ (* grasetorigin fija la situacion del origen de la ventana y
292@    mueve la posicion corriente a ese punto. *)
293@
294@ PROCEDURE grasetorigin(x,y:integer);
295@ BEGIN
296@  rde:=x; rhl:=y;
297@  user(#bbc9)
```

```
298@ END;
299@
300@ (* gragetorigin entrega la posicion del origen de ventana. *)
301@
302@ PROCEDURE gragetorigin(VAR x,y:integer);
303@ BEGIN
304@  user(#bbcc);
305@  x:=rde; y:=rhl
306@ END;
307@
308@ (* grawinwidth fija la anchura de la ventana de graficos,
309@    dando los bordes derecho e izquierdo. *)
310@
311@ PROCEDURE grawinwidth(x1,x2:integer);
312@ BEGIN
313@  rde:=x1; rhl:=x2;
314@  user(#bbcf)
315@ END;
316@
317@ (* grawinheight fija la altura de ventana, dandole los bordes
318@    superior e inferior. *)
319@
320@
321@ PROCEDURE grawinheith(y1,y2:integer);
322@ BEGIN
323@  rde:=y1; rhl:=y2;
324@  user(#bbd2)
325@ END;
326@
327@ (* gragetwwidth entrega la anchura de ventana, dando como resultado los
328@    valores del borde izquierdo y derecho de ella. *)
329@
330@ PROCEDURE gragetwwidth(VAR x1,x2:integer);
331@ BEGIN
332@  user(#bbd5);
333@  x1:=rde; x2:=rhl
334@ END;
335@
336@ (* gragetwheight entrega la altura de ventana, dando como resultado los
337@    valores del borde superior e inferior. *)
338@
339@ PROCEDURE gragetwheight(VAR y1,y2:integer);
340@ BEGIN
341@  user(#bbd8);
342@  y1:=rde; y2:=rhl
343@ END;
344@
345@ (* graclearwindow limpia la ventana de graficos. *)
346@
```

```
347@ PROCEDURE graclearwindow;
348@ BEGIN
349@  user(#bbdb)
35@@ END;
351@
352@ (* grasetpen fija la tinta que corresponde a la pluma de graficos,
353@    que sera la usada para dibujar puntos y lineas. *)
354@
355@ PROCEDURE  grasetpen(INK:integer);
356@ BEGIN
357@  ra:=chr(INK);
358@  user(#bbde)
359@ END;
36@@
361@ (* gragetpen entrega el numero de tinta que corresponde a la pluma. *)
362@
363@ FUNCTION gragetpen:integer;
364@ BEGIN
365@  user(#bbe1);
366@  gragetpen:=ord(ra)
367@ END;
368@
369@ (* grasetpaper fija la tinta que corresponde al papel, que es el color
37@@    usado para el fondo de la pantalla. *)
371@
372@ PROCEDURE grasetpaper(INK:integer);
373@ BEGIN
374@  ra:=chr(INK);
375@  user(#bbe4)
376@ END;
377@
378@ (* gragetpaper entrega la cinta del papel corriente. *)
379@
38@@ FUNCTION gragetpaper:integer;
381@ BEGIN
382@  user(#bbe7);
383@  gragetpaper:=ord(ra)
384@ END;
385@
386@ (* graplotabsolute pinta un punto en una posicion dada por sus
387@    coordenadas absolutas,usando la tinta de pluma corriente y el modo de
388@    escritura de graficos vigente. *)
389@
39@@ PROCEDURE graplotabsolute(x,y:integer);
391@ BEGIN
392@  rde:=x;rhl:=y;
393@  user(#bbea)
394@ END;
```

```
3950
3960 (* graplotrelative pinta un punto en la posicion dada por las coordenadas
3970    relativas a la posicion corriente, usando la tinta de pluma corriente
3980    y en el moo de escritura de graficos vigente. *)
3990
4000 PROCEDURE graplotrelative(x,y:integer);
4010 BEGIN
4020  rde:=x;rhl:=y;
4030  user(#bbed)
4040 END;
4050
4060
4070 (* gratestabsolute mueve la posicion corriente a la posion mencionada
4080    y devulve el valor de la tinta que encuentre en ese punto. *)
4090
4100 FUNCTION gratetabsolute(x,y:integer):integer;
4110 BEGIN
4120  rde:=x; rhl:=y;
4130  user(#bbf0);
4140  gratestabsolute:=ord(ra)
4150 END;
4160
4170 (* gratestrelative mueve el puntero de graficos a la posicion mencionada
4180    en cooredenadas relativas a la posion corriente,y entrega de la
4190    tinta que encuentre en ese punto. *)
4200
4210 FUNCTION gratestrelative(x,y:integer):integer;
4220 BEGIN
4230  rde:=x; rhl:=y;
4240  user(#bbf3);
4250  gratestrelative:=ord(ra)
4260 END;
4270
4280 (* gralineabsolute mueve la posicion corriente hasta el punto final de
4290    la linea mencionada, trazando una linea desde la posicion corriente
4300    y usando la tinta corriente, y el modo de escritura de graficos
4310    vigente. *)
4320 PROCEDURE gralineabsolute(x,y:integer);
4330 BEGIN
4340  rde:=x; rhl:=y;
4350  user(#bbf6)
4360 END;
4370
4380 (* gralinerelative mueve la posicion corriente hasta el punto final
4390    mencionado por sus coordenadas relativas,y traza una linea hasta el
4400    desde la posicion corriente, y usando la tinta y modo de graficos
4410    corriente. *)
4420 PROCEDURE gralinerelative(x,y:integer);
4430 BEGIN
```

```
444@  rde:=x; rhl:=y;
445@  user(#bbf9)
446@ END;
447@
448@ (* grawrchar escribe un caracter en la posicion corriente del cursor de
449@    graficos, y mueve dicha posicion a la derecha preparandose para
45@@    escribir dicho caracter. *)
451@
452@ PROCEDURE grawrchar(c:char);
453@ BEGIN
454@  ra:=c;
455@  user(#bbfc)
456@ END;
457@
458@ (* scrinitialise restaura los valores iniciales del paquete de pantalla,
459@    el modo, y las tintas a usar. *)
46@@
461@ PROCEDURE scrinitialise;
462@ BEGIN
463@  user(#bbff)
464@ END;
465@
466@ (* scrsetoffset fija el 'desplazamiento' del primer caracter de la
467@    pantalla. Cambiando este desplazamiento se puede conseguir que sean
468@    los circuitos los que hagan los 'corrimientos' de imagen. *)
469@
47@@ PROCEDURE scrsetoffset(INK:integer);
471@ BEGIN
472@  rhl:=INK;
473@  user(#bc@5)
474@ END;
475@
476@ (* scrgetlocation entrega el desplazamiento estipulado en ese momento
477@    para el primer caracter de la pantalla. *)
478@
479@ FUNCTION scrgetlocation:integer;
48@@ BEGIN
481@  user(#bc@b);
482@  scrgetlocation:=rhl
483@ END;
484@
485@ (* scrsetmode fija la pantalla a un modo determinado; limpia la pantalla
486@    y estipula la ventana como toda la pantalla. *)
487@
488@ PROCEDURE scrsetmode(m:integer);
489@ BEGIN
49@@  ra:=chr(m);
491@  user(#bc@e)
492@ END;
493@
```

```
494@ (* scrgetmode entrega al modo de pantalla corriente. *)
495@
496@ FUNCTION scrgetmode:integer;
497@ BEGIN
498@  user(#bc11);
499@  scrgetmode:=ord(ra)
500@ END;
501@
502@ (* scrclear limpia la pantalla dejandola en tinta 0. *)
503@
504@ PROCEDURE scrclear;
505@ BEGIN
506@  user(#bc14)
507@ END;
508@
509@ (* scrcharlimits entrega el limite para la ultima columna de caracteres
510@    y la ultima fila para la pantalla en el modo vigente. *)
511@
512@ PROCEDURE scrcharlimits(VAR col,row:integer);
513@ BEGIN
514@  user(#bc17);
515@  col:=ord(rb); row:=ord(rc)
516@ END;
517@
518@ (* scrsetink fija los colores que corresponden a latinta mencionada.Si
519@    los dos colores son el mismo, la tinta mostrara un solo color. Si
520@    son distintos, la tinta alternara entre los
521@    dos colores dados. *)
522@
523@ PROCEDURE scrsetink(INK,col1,col2:integer);
524@ BEGIN
525@  ra:=chr(INK); rb:=chr(col1); rc:=chr(col2);
526@  user(#bc32)
527@ END;
528@
529@ (* scrgetink entrega los dos colores que estan asignados
530@    a la tinta mencionada. *)
531@
532@ PROCEDURE scrgetink(VAR col1,col2:integer);
533@ BEGIN
534@  user(#bc35);
535@  col1:=ord(rb); col2:=ord(rc)
536@ END;
537@
538@ (* scrsetborder fija los dos colores a usar para pintar el reborde
539@    de la pantalla. Si los dos colores son el mismo, el reborde tendra un
540@    solo color, si son distintos, parpadeara entre ellos. *)
541@
542@ PROCEDURE scrsetborder(col1,col2:integer);
```

```
543@ BEGIN
544@  rb:=chr(col1); rc:=chr(col2);
545@  user(#bc38)
546@ END;
547@
548@ (* scrgetborder entrega los dos colores estipulados para pintar el
549@    reborde de la pantalla en ese momento. *)
55@@
551@ PROCEDURE scrgetborder(VAR col1,col2:integer);
552@ BEGIN
553@  user(#bc3b);
554@  col:=ord(rb); col2:=ord(rc)
555@ END;
556@
557@ (* scrsetflashing fija la duracion del parpadeo, o alternancia entre dos
558@    colores que tendran las tintas que aparecen en pantalla. Estos valores
559@    estipulados se aplican para la pluma, el papel, y el reborde. El
56@@    periodo de parpadeo se da en 1/5@ sg. (1/6@ sg. en USA).
561@    El valor prescrito
562@    es de 1@. *)
563@
564@ PROCEDURE scrsetflashing(p1,p2:integer);
565@ BEGIN
566@  rb:=chr(p1); rc:=chr(p2);
567@  user(#bc3e)
568@ END;
569@
57@@ (* scrgetflashing entrega el valor estipulado para el periodo
571@    de parapadeo, tal como se ha descrito en el anterior procedimiento. *)
572@
573@ PROCEDURE scrgetflashing(VAR p1,p2:integer);
574@ BEGIN
575@  user(#bc41);
576@  p1:=ord(rb); p2:=ord(rc)
577@ END;
578@
579@ (* scrfillbox rellena el cuadradito ocupado por un caracter en la pantalla
58@@    con la tinta mencionada. Col1 y col2 son las columnas izquierda y
581@    derecha del area que se rellena.Row1 y row2 son la 'fila' superior e
582@    inferior de dicho area. Las coordenadas son 'fisicales'. *)
583@
584@ PROCEDURE scrfillbox(INK,col1,col2,row1,row2:integer);
585@ BEGIN
586@  ra:=chr(INK);
587@  rh:=chr(col1); rd:=chr(col2);
588@  rl:=chr(row1); re:=chr(row2);
589@  user(#bc44)
59@@ END;
591@
592@ (* scrcharinvert invierte los colores usados para pintar un caracter.
593@    Todos los elementos de imagen en la posicion donde esta escrito el
```

```
594@    caracter cambian su tinta-video inverso. Las coordenadas usadas son
595@    coordenadas 'fisicales', *)
596@
597@ PROCEDURE scrcharinvert(i1,i2,col,row:integer);
598@ BEGIN
599@  rb:=chr(i1); rc:=che(i2);
6@@@  rh:=chr(col); rl:=chr(row);
6@1@  user(#bc4a)
6@2@ END;
6@3@
6@4@ (* scrhwroll corre toda la imagen hacia arriba o hacia abajo 8 puntos
6@5@    (un caracter), y el corrimiento lo efectuan los circuitos. La linea
6@6@    en blanco se rellena con el color de tinta mencionado como parametro*)
6@7@
6@8@ PROCEDURE scrhwroll(up:boolean; INK:integer);
6@9@ BEGIN
61@@  rb:=chr(ord(up)); ra:=chr(INK);
611@  user(#bc4d)
612@ END;
613@
614@ (* scrswroll corre un area de la pantalla hacia arriba o hacia abajo
615@    8 puntos (un caracter), usando los propios programas. El area de
616@    coordenadas se da en coordenadas absolutas. *)
617@
618@ PROCEDURE scrswroll(up:boolean;INK,col1,col2,row1,row2:integer);
619@ BEGIN
62@@  rb:=chr(ord(up)); ra:=chr(INK);
621@  rh:=chr(col1); rd:=chr(col2);
622@  rl:=chr(row1); re:=chr(row2);
623@  user(#bc5@)
624@ END;
625@
626@ (* scracces fija el modo de escritura en pantalla para la VDU de graficos
627@    Los valores posibles para el modo:
628@    @: modo FORCE  NEW= INK
629@    1: modo XOR    NEW= INK oleado por OLD
63@@    2: modo AND    NEW= INK yliado por OLD
631@    3: modo OR     NEW= INK oliado por OLD
632@
633@   NEW es la estipulacion final de tinta para el elemento.
634@   OLD es la estipulacion corriente de tinta para el elemento.
635@   INK es la tinta  estipulada para pintar.
636@
637@   El modo prescrito para omisiones es el modo forzado-modo @. *)
638@
639@ PROCEDURE scracces(m:integer);
64@@ BEGIN
641@  ra:=chr(m);
642@  user(#bc59)
```

```
643@ END;
644@
645@ (* scrhorizontal pinta una simple linea horizontal entre dos puntos
646@    usando el modo de escritura vigente para graficos. *)
647@
648@ PROCEDURE scrhorizontal(INK,x1,x2,y:integer);
649@ BEGIN
650@  ra:=chr(INK);
651@  rde:=x1; rbc:=x2; rhl:=y;
652@  user(#bc5f)
653@ END;
654@
655@ (* scrvertical pinta una simple linea vertical entre dos puntos usando
656@    el modo de escritura para graficos vigente. *)
657@
658@ PROCEDURE scrvertical(INK,x,y1,y2:integer);
659@ BEGIN
660@  ra:=chr(INK);
661@  rde:=x; rhl:=y1; rbc:=y2;
662@  user(#bc62)
663@ END;
664@
665@ (* soundreset restaura las condiciones iniciales en el generador de sonido
666@    No se generaran mas notas y se vaciaran todos los 'chorros'pendientes*)
667@
668@ PROCEDURE soundreset;
669@ BEGIN
670@  user(#bca7)
671@ END;
672@
673@ (* soundhold detiene todas las notas en medio de su emision. El resultado
674@    de esta funcion es cierto, si habia una nota activa. *)
675@
676@ FUNCTION soundhold:boolean;
677@ BEGIN
678@  user(#bcb6);
679@  soundhold:=odd(raf)
680@ END;
681@
682@ (* soundcontinue reanuda la emision de notas despues de que hayan sido
683@    'retenidas' por la funcion anterior. *)
684@
685@ PROCEDURE soundcontinue;
686@ BEGIN
687@  user(#bcb9)
688@ END;
689@
690@ (* soundaaddrress entrega la direccion de la envolvente de la amplitud
691@    correspondiente al parametro dado. *)
692@
693@ FUNCTION soundaaddress(e:integer):integer;
```

```
694@ BEGIN
695@  ra:=chr(e);
696@  user(#bcc2);
697@  soundaaddress:=rhl
698@ END;
699@
7@@@ (* soundtaddress entrega la direccion de todo correspondiente al
7@1@    parametro dado. *)
7@2@
7@3@ FUNCTION soundtaddress(e:integer):integer;
7@4@ BEGIN
7@5@  ra:=chr(e);
7@6@  user(#bcc5);
7@7@  soundtaddress:=rhl
7@8@ END;
7@9@
71@@ (* mcprintchar intenta enviar un carácter hacia el 'portico' Centronics.
711@    La funcion entrega el valor cierto si puede enviarlo; en los
712@    demas casos entrega falso despues de @,4 segundos. *)
713@
714@ FUNCTION mcprintchar(c:char):boolean;
715@ BEGIN
716@  ra:=c;
717@  user(#bd2b);
718@  mcprintchar:=odd(raf)
719@ END;
72@@
721@ (* mcbusyprinter entrega cierto si la via de ocupado en el 'portico'
722@    Centronics esta 'busy', en caso contrario, entrega falso. *)
723@
724@ FUNCTION mcbusyprinter:boolean;
725@ BEGIN
726@  user(#bd2e);
727@  mcbusyprinter:=odd(raf)
728@ END;
729@
```

AMSTRAD