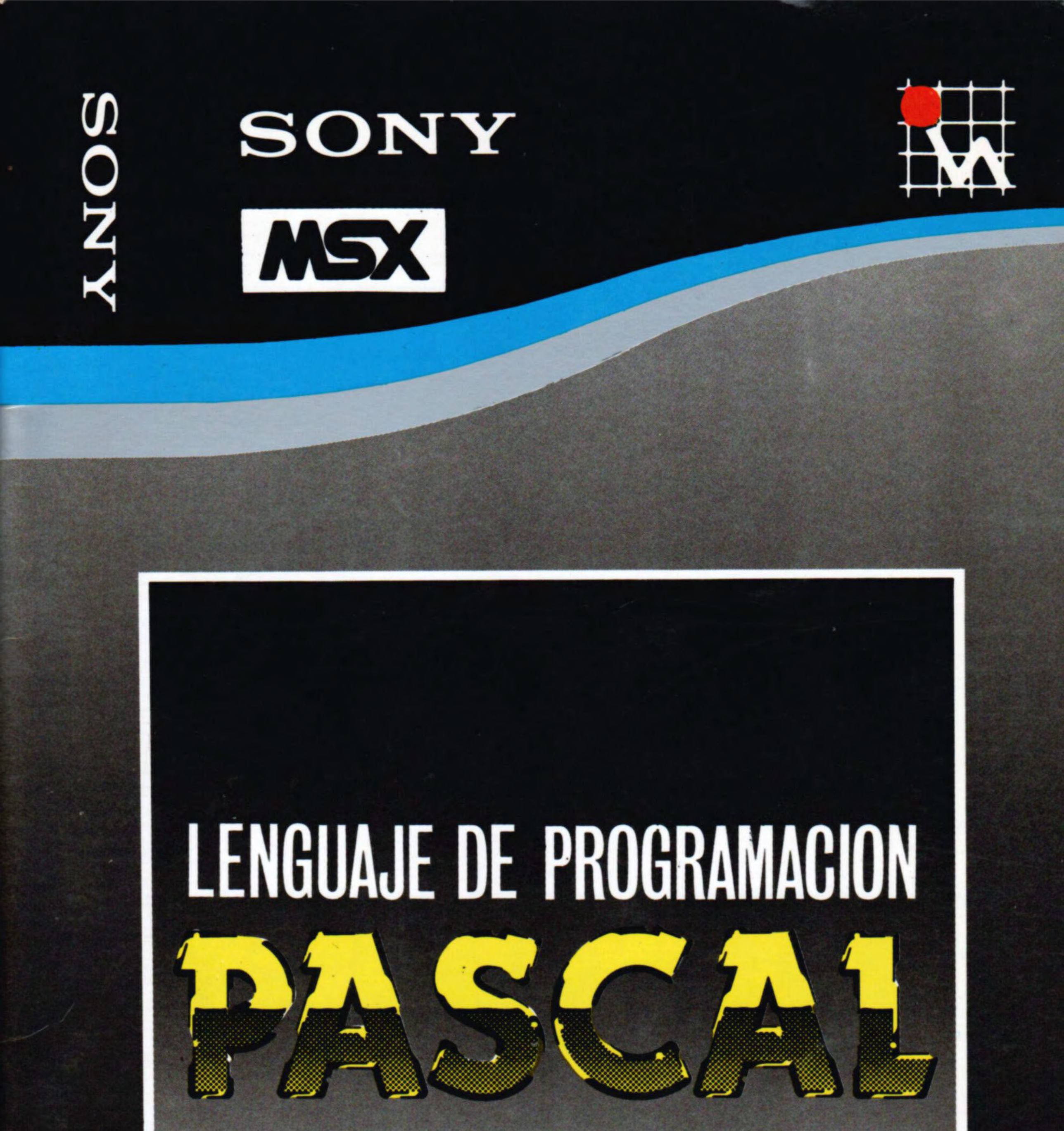
SONY

SONY

MSX



LENGUAJE DE PROGRAMACION

PASCAL

CONTENIDO

PASCAL

Edición original: HISOFT. Primera edición: 1984.
Edición española: INDESCOMP, S.A.

HISOFT, 180 High Street, North Dunstable LU6 1AT
Indescomp, S.A., Avda. del Mediterráneo 9, 28007 Madrid

Producción de la edición española:
Vector Ediciones, Gutierre de Cetina 61, 28017 Madrid

Traducción:
Jesús Rojo García
Profesor de Matemática Aplicada
Escuela T. Sup. de Ing. Industriales de Valladolid

Impresión:
Gráficas Lormo, Isabel Méndez 15, 28038 Madrid

CONTENIDO

SECCIÓN Ø. INTRODUCCIÓN

Comenzaremos por felicitarle por ser propietario de esta implementación casi completa del lenguaje de programación Pascal que es el HISOFT PASCAL. Podrá usted comprobar que se trata de un potente instrumento para escribir programas bien estructurados y fáciles de entender. Como ocurre con cualquier lenguaje de programación, utilizar Hisoft Pascal le costará cierto esfuerzo al principio.

Lo que está usted leyendo es un manual de referencia, es decir, un libro en el que se detallan los aspectos particulares de Hisoft Pascal. El libro no está pensado para que usted aprenda Pascal; si no conoce nada de Pascal, le recomendamos que lea también alguno de los libros que figuran en la bibliografía de las últimas páginas.

Si es la primera vez que consulta este libro, es aconsejable que se atenga al siguiente orden:

—lea el resto de esta sección Ø y desarrolle los ejemplos que le proponemos para que escriba, compile y ejecute;
—lea la sección dedicada al editor (la 4) y ensaye con el ejemplo que figura al final de dicha sección;
—insista en lo anterior hasta que se sienta seguro en el uso del editor y en la compilación y ejecución de un programa en Pascal;
—si en algún momento se encuentra bloqueado, deje un rato el ordenador, haga cualquier otra cosa y vuelva a comenzar otra vez; es bastante normal que ocurra esto siempre que se entra en algo completamente nuevo;
—si a pesar de todo piensa que no es usted el responsable de las dificultades que encuentra, no dude en ponerse en contacto con Hisoft; nuestro equipo está dispuesto a responder a las preguntas sobre nuestros productos.

Para los que ya conocen algo de Pascal, vamos a hacer algunas precisiones. Hisoft Pascal (que abreviaremos con HP) es una versión potente, sencilla y rápida del Pascal normalizado que se describe en el libro de Jensen y Wirth, "Pascal User Manual and Report" que citamos en la bibliografía. Adolece de algunas omisiones y posee ciertas características adicionales:

—No están implementados los ficheros (FILE), si bien se pueden almacenar las variables en cinta.
—Un tipo RECORD puede no tener una parte VARIANT.
—PROCEDURES y FUNCTIONS no son válidos como parámetros.
—Incluye procedimientos y funciones suplementarios para aprovechar el entorno particular en el que se utiliza el compilador; entre éstas se encuentran POKE, PEEK, TIN, TOUT y ADDR.

El compilador ocupa unos 12K, las rutinas de ejecución 4K y el editor 2K; la implementación ocupa entonces 18K y deja el resto de la memoria para los propios programas en Pascal y los programas objeto (unos 41K en los ordenadores de 64K).

Ø.1 Cómo se carga Hisoft Pascal

Para cargar HP, introduzca la cinta en el magnetófono, de manera que la etiqueta 'Hisoft Pascal' quede en la parte superior, y rebobine la cinta. Escriba

RUN "CAS:HPMSX" ⟨RETURN⟩

y ponga el magnetófono en posición de lectura. Cuando el cargador BASIC haya terminado, emitirá el mensaje 'Hisoft Pascal MSX Loader' y se cargará la parte que está en código de máquina.

Entonces, el mensaje

Stack Address:-([RETURN] is default)?

aparecerá en pantalla. Normalmente bastará con que apriete ⟨RETURN⟩. Sin embargo, es posible que desee tener cargada alguna rutina en código de máquina simultáneamente al compilador; en ese caso deberá cargarla por debajo del comienzo del compilador (en la posición hexadecimal #A7ØØ) e introducir, en respuesta al mensaje, la posición de memoria más baja que utiliza esta rutina (hay que introducirla en forma decimal).

El control pasará entonces al editor, que enviará a la pantalla un panel de ayudas y esperará a que usted le introduzca algún comando.

Ø.2 Desarrolle usted mismo un ejemplo

Ya puede usted comenzar con algún programa. Nosotros le proponemos desarrollar un par de ejemplos muy sencillos. Como tendrá que comenzar por escribirlos, será conveniente que le digamos que puede ayudarse de los comandos de edición del MSX, que son, además de las flechas del cursor, los siguientes:

⟨CTRL/B⟩	mueve el cursor a la palabra anterior,
⟨CTRL/C⟩	vuelve al modo comando,
⟨CTRL/E⟩	borra hasta el final de la línea,
⟨CTRL/F⟩	lleva el cursor al comienzo de la siguiente palabra,
⟨BS⟩	borra el carácter que precede al cursor,
⟨HOME⟩	lleva el cursor a la esquina superior izquierda de la pantalla,
⟨SHIFT/HOME⟩	borra completamente la pantalla,
⟨RETURN⟩	valida la línea,
⟨CTRL/N⟩	lleva el cursor al final de la línea,
⟨INS⟩	cambia del modo inserción o sobreescritura a su contrario,
⟨CTRL/U⟩	borra la línea,
⟨DELETE⟩	borra el carácter sobre el que está el cursor.

Recuerde que ninguna línea queda introducida hasta que se pulsa ⟨RETURN⟩.

Bien, vamos a escribir el primer programa; teclee

I1Ø,1Ø⟨RETURN⟩

y el editor le invitará a escribir la línea número 1Ø. Cuando la haya introducido, le invitará a continuar con la línea 2Ø, y así hasta que usted pulse ⟨CTRL/STOP⟩. Escriba así el siguiente programa

```
10 PROGRAM HOLA;
20 BEGIN
30  WRITELN('HOLA BUENOS DIAS!');
40 END.
50 <CTRL/STOP>
```

donde ⟨CTRL/STOP⟩ no aparecerá en su pantalla; nosotros lo ponemos para recordarle que debe pulsarlo. Observe que hay un punto (importante) después de 'END' en la línea 40.

Ahora, para compilar este programa, pulse

C⟨RETURN⟩

Verá aparecer un listado de su programa con algunos números extra al principio; es un listado de com-

pilación. Si el programa se ha compilado sin problemas, aparecerá el mensaje 'Run?' (ejecución?); responda pulsando 'Y' y entonces el programa se ejecutará, escribirá en la pantalla

HOLA BUENOS DIAS!

y el control volverá de nuevo al editor. Si desea ejecutar el programa de nuevo, pulse

R⟨RETURN⟩

Si el programa no se compila correctamente y aparece algún mensaje de '*ERROR*', vuelva al editor pulsando

E⟨RETURN⟩

luego liste el programa con

L⟨RETURN⟩

y compárelo con el que quería escribir. Si se ha confundido en alguna línea, pulse el número de la línea, deje un espacio en blanco, escríbala de nuevo e introdúzcala con ⟨RETURN⟩. Pruebe a compilar de nuevo y a corregir, si es necesario, hasta que tenga éxito.

Ahora vamos a probar con otro programa. Borre previamente el anterior utilizando

D1,9999⟨RETURN⟩

y escriba a continuación el nuevo programa:

```
10  PROGRAM CARASCII
20  VAR CA : CHAR;
30  BEGIN
50   REPEAT
40    WRITE('INTRODUZCA UN CARACTER ');
60    READLN;
70    READ(CA);
80    WRITELN(CA,' ES ',ORD(CA),' EN ASCII.');
90   UNTIL CA=' ';
100 END.
110 <CTRL/STOP>
```

que podrá compilar y ejecutar (siga las instrucciones del ejemplo anterior).

Al ejecutarlo, el programa le pedirá que 'INTRODUZCA UN CARACTER'; pulse uno cualquiera, seguido de ⟨RETURN⟩, y el programa le escribirá el carácter pulsado y a su lado el número ASCII correspondiente. Esto se repetirá hasta que usted teclee un espacio como respuesta.

Convendrá que observe, si conoce algo de Pascal, la utilización de READLN y de READ en este ejemplo, y que amplie el estudio con la sección 2.3.1.

Con estos ejemplos ha podido hacerse una idea de cómo funciona HP. Lea el resto de la sección y pase después a la sección 4.

Ø.3. Compilar y ejecutar

Aunque la mayor parte de las precisiones sobre la creación, corrección, compilación y ejecución de programas se proporcionan en la sección 4, vamos a comentar aquí algunos detalles al respecto.

Cuando se llama al compilador, se genera un listado en el que cada línea es de la forma

XXXX nnnn texto de la línea

donde XXXX es la dirección en que comienza el código objeto generado por la línea y nnnn es el número de línea (sin los ceros iniciales). El listado puede salir por la pantalla o por la impresora, dependiendo del uso que se haga de la opción 'P' de compilación. El listado se detiene pulsando ⟨CTRL/STOP⟩; a continuación se puede volver al editor pulsando ⟨CTRL/STOP⟩ de nuevo, o se puede proseguir el listado si se pulsa cualquier otra tecla.

Si se detecta un error durante la compilación, entonces aparece el mensaje '*ERROR*', seguido de la flecha ^ apuntando justo *después* del símbolo culpable del error y, finalmente, un número de error (el significado de los números de error se explica en el apéndice 1); además, el listado se detiene. Entonces se puede pulsar 'E' para editar la última línea, 'P' para editar la línea anterior (si existe) o cualquier otra tecla para proseguir la compilación.

Si el programa no termina con 'END.' entonces el mensaje de error toma la forma 'No more text' (no más texto) y el control pasa al editor.

Si el compilador desborda el espacio reservado para la tabla de símbolos del compilador, se produce el mensaje de error 'No Table Space' y el control pasa al editor. Entonces se debe utilizar el comando 'a' (véase la sección 4) para reservar un espacio mayor en respuesta a la pregunta 'Symbol Table Size (Ø5ØØ)?' sobre el tamaño de dicha tabla.

Si al terminar la compilación se han producido errores, el compilador mostrará en pantalla el número de errores detectados y destruirá el código objeto. Si no han existido errores, aparecerá el mensaje 'Run' (ejecución). Si la respuesta es 'Y' el programa se ejecuta; en caso contrario, el control pasa al editor.

Durante la ejecución se pueden detectar errores, lo que provocará la aparición de mensajes de error (véase el apéndice 1).

La ejecución de un programa se interrumpe pulsando ⟨CTRL/STOP⟩. A continuación, pulsando de nuevo ⟨CTRL/STOP⟩, se deja definitivamente sin efecto la ejecución; pulsando cualquier otra tecla, se reanuda la ejecución.

Ø.4. El resto del manual

—La sección 1 proporciona la sintaxis y la semántica que espera el compilador.

—La sección 2 describe los identificadores predefinidos en Hisoft Pascal.

—La sección 3 explica las distintas opciones de compilación y también el formato de los comentarios.

—La sección 4 enseña el uso del editor de línea de que está dotado HP.

—El apéndice 1 da la lista de los números y mensajes de error que pueden ocurrir durante la compilación y la ejecución.

—El apéndice 2 da la lista de las palabras reservadas y los identificadores predefinidos.

—El apéndice 3 explica la representación interna de los distintos tipos de datos, para quienes gustan de meterse en profundidades.

—El apéndice 4 proporciona algunos ejemplos de programas; estúdielos si se le presentan problemas a la hora de escribir programas.

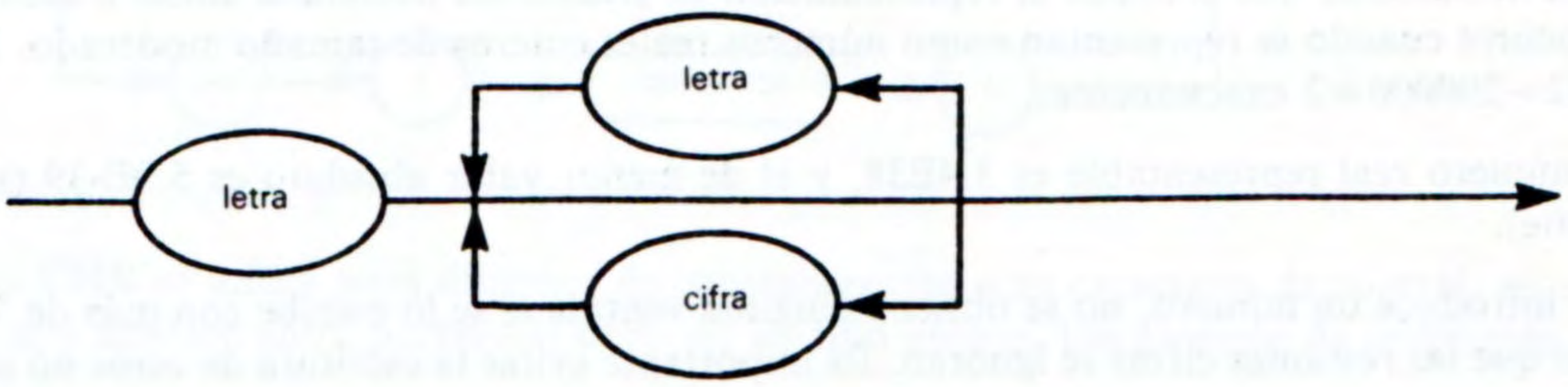
Ya hemos dicho que lo más aconsejable es continuar ahora con la sección 4.

SECCIÓN 1. SINTAXIS Y SEMÁNTICA

En esta sección describimos la sintaxis y la semántica de Hisoft Pascal, que corresponde a la especificada en la segunda edición del "Pascal User Manual and Report" de K. Jensen y N. Wirth (véase la bibliografía).

Añadimos también los diagramas sintácticos, que ayudan a comprender las reglas de trabajo del compilador.

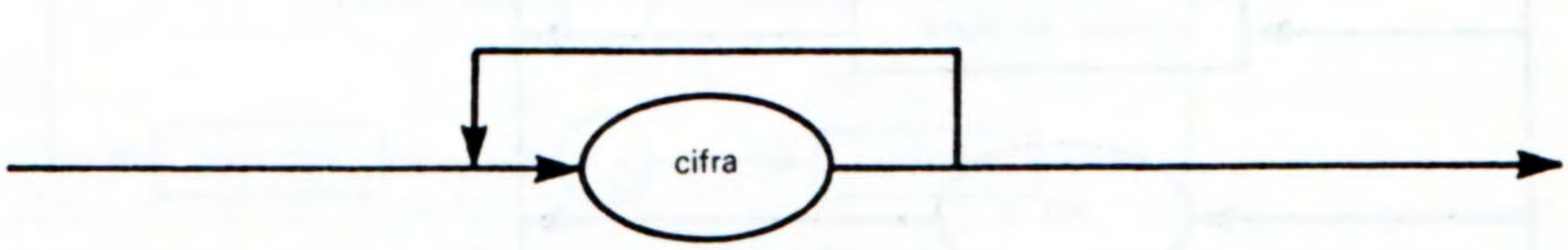
1.1 Identificador



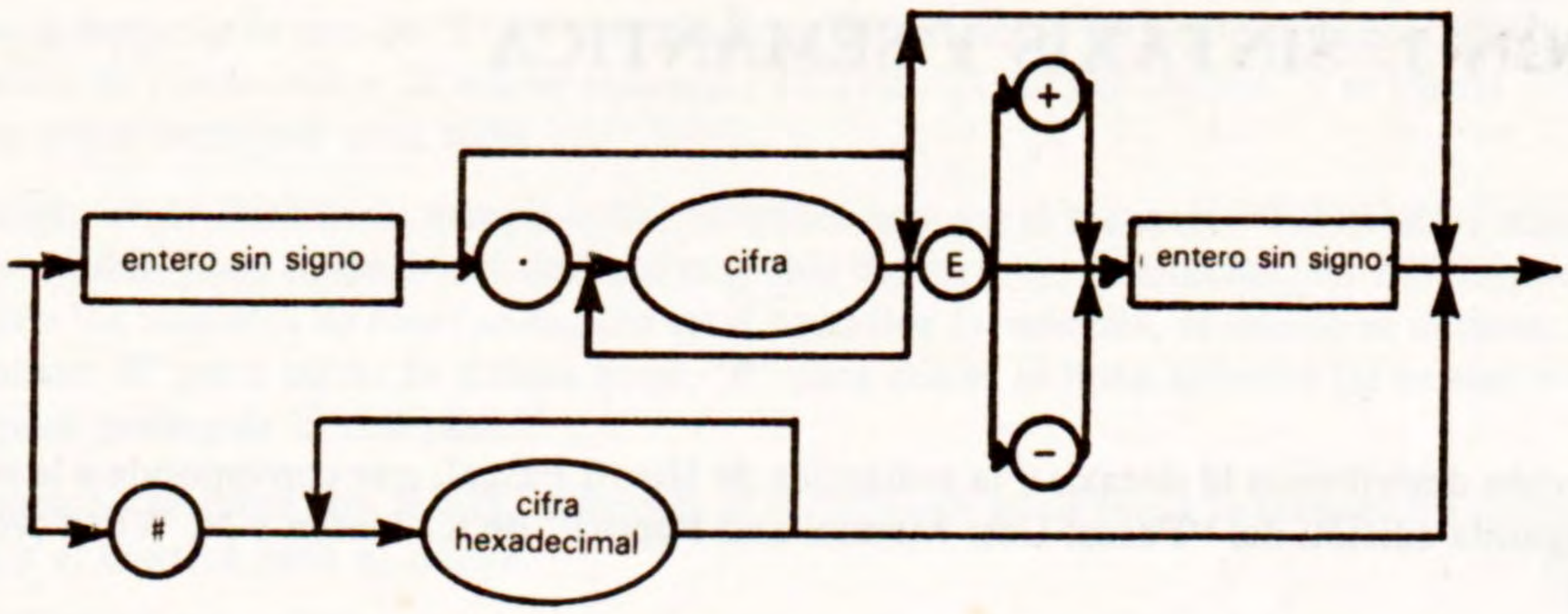
Sólo se consideran significativos los 1Ø primeros caracteres de cada identificador.

Los identificadores pueden contener letras minúsculas y mayúsculas, pero no son equivalentes unas y otras. Por ejemplo, HOLA, HOla y hola son identificadores diferentes. Las palabras reservadas y los identificadores predefinidos deben ser tecleados obligatoriamente en mayúsculas.

1.2 Entero sin signo



1.3 Número sin signo



Para HP los enteros tienen un valor absoluto menor o igual que 32767; todo número mayor se trata como real.

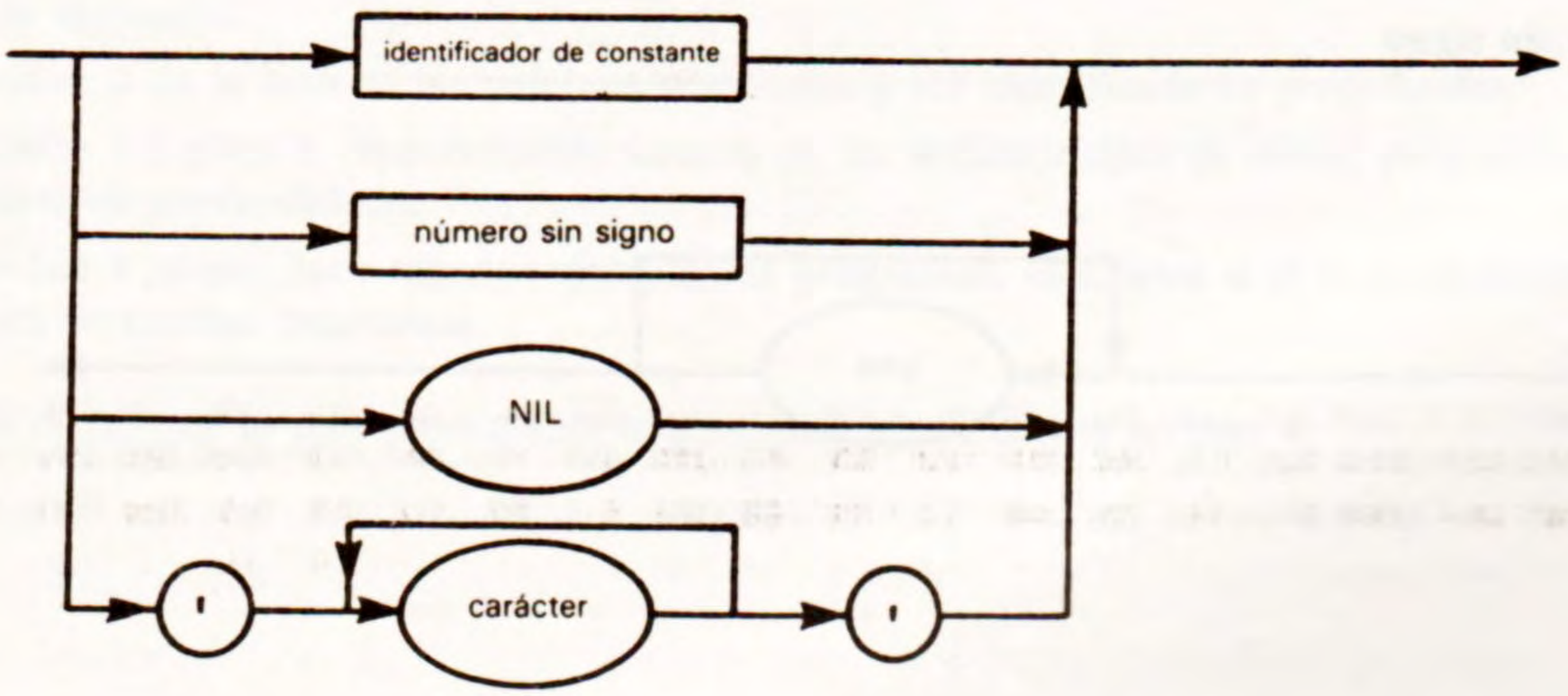
La mantisa de los números reales es de una longitud de 23 bits; esto equivale a trabajar con unas 7 cifras significativas. Obsérvese que se pierde exactitud cuando se realiza una operación cuyo resultado es mucho menor que el valor absoluto de los argumentos. Por ejemplo, 2.ØØØØ2-2 no da Ø.ØØØØ2. Esto se debe a la inexactitud que provoca la representación de fracciones decimales como fracciones binarias, y no ocurre cuando se representan como números reales enteros de tamaño moderado. Por ejemplo, 2ØØØØ2 – 2ØØØØØ = 2 exactamente.

El mayor número real representable es 3.4E38, y el de menor valor absoluto es 5.9E-39 (salvo el Ø, naturalmente).

Cuando se introduce un número, no se obtiene ninguna ventaja si se lo escribe con más de 7 cifras de mantisa, ya que las restantes cifras se ignoran. Es importante evitar la escritura de ceros no significativos, puesto que éstos contarán en el número de cifras. Por ejemplo, Ø.ØØØ123456 se representará con menor exactitud que 1.23456E-4.

Los números hexadecimales tienen interés, entre otras cosas, para especificar direcciones de memoria. Después del símbolo '#' debe haber al menos una cifra hexadecimal; en caso contrario se genera el mensaje '*ERROR*51'

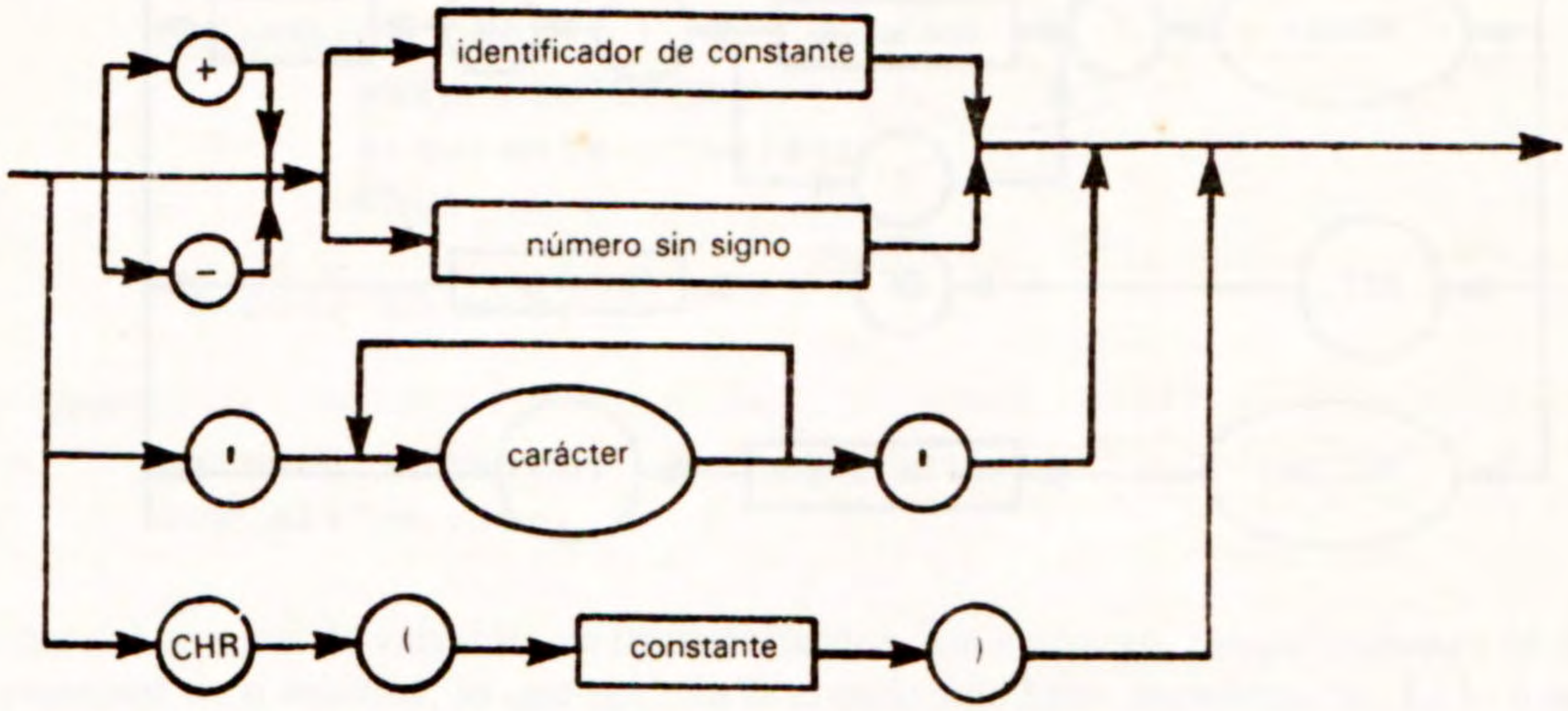
1.4 Constante sin signo



Los caracteres posibles son los 256 del conjunto de caracteres ASCII. Con objeto de mantener la compatibilidad con el Pascal normalizado, el carácter nulo no se puede introducir como '', sino como CHR(Ø).

Las cadenas literales están limitadas a 255 caracteres. Deben ser declaradas de tipo ARRAY [1..N] OF CHAR, donde N es un entero entre 1 y 255. No deben contener el carácter de fin de línea, CHR(13), so pena de generar el '*ERROR*68'.

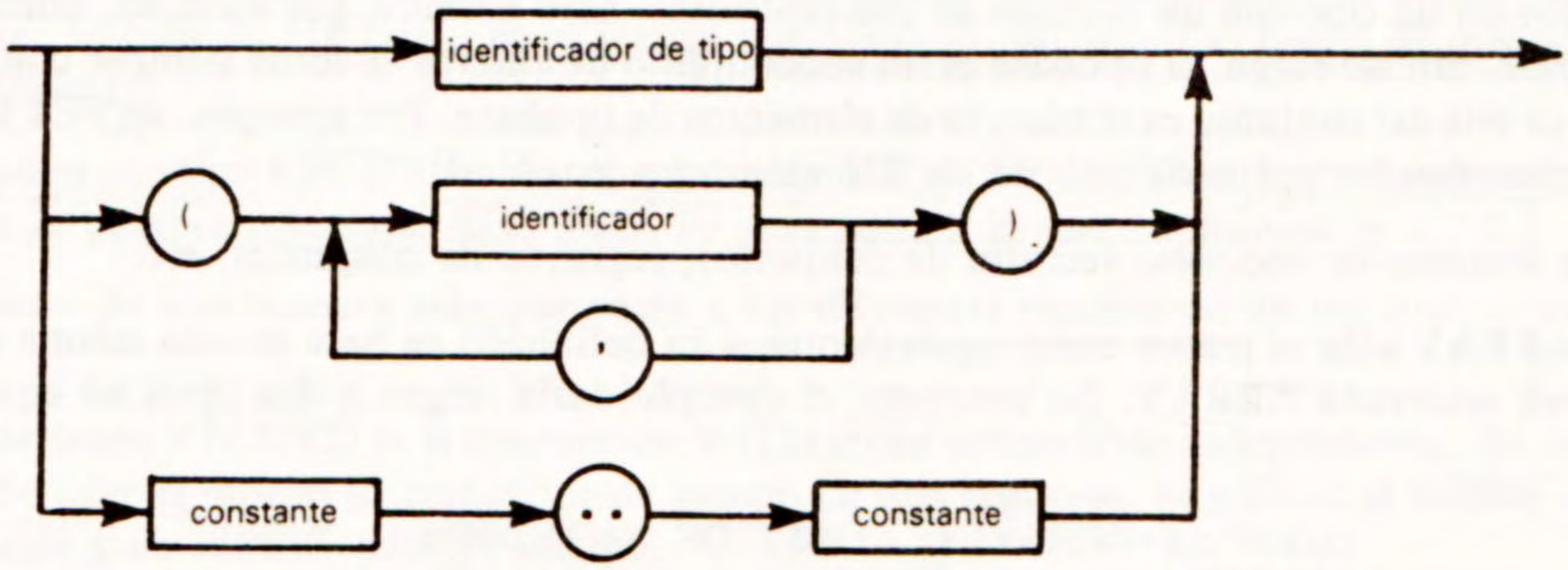
1.5 Constante (CONST)



La función CHR se utiliza para disponer de constantes que sean caracteres de control; en ese caso, la constante que aparece entre paréntesis debe ser de tipo entero. Un ejemplo de utilización puede ser

```
CONST bs=CHR(8);
      cr=CHR(13);
```

1.6 Tipo simple (TYPE)

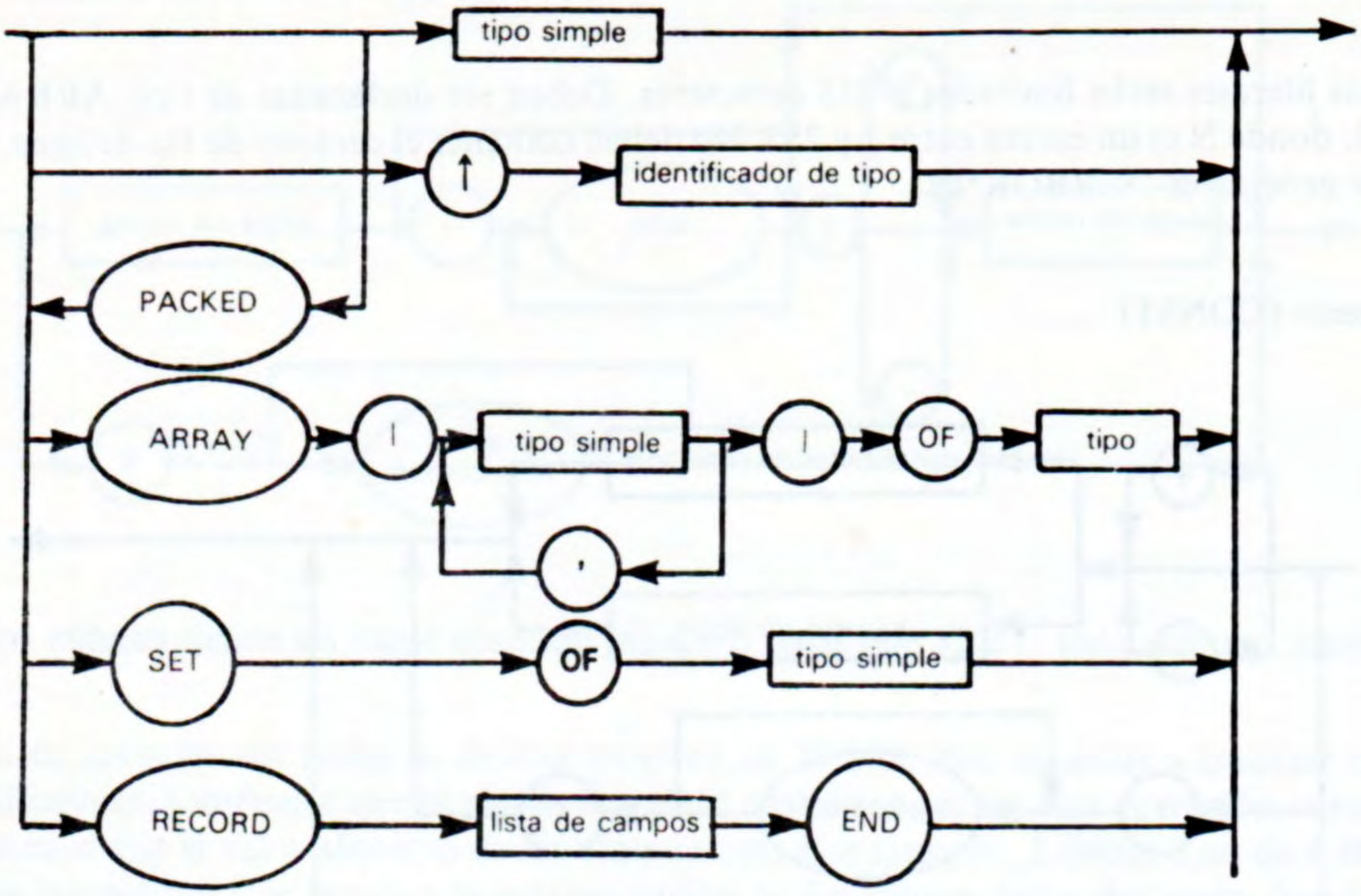


Los tipos declarados por enumeración

TYPE ident = (ident,ident,...)

no deben tener más de 256 elementos.

1.7 Tipo (TYPE)



La palabra PACKED se acepta, aunque no tiene ningún efecto, ya que el empaquetamiento tiene lugar de manera natural para un vector (ARRAY) de caracteres, o de otro tipo. La única circunstancia en que el empaquetamiento de un vector podría presentar ventajas es el caso de tipo lógico (BOOLEAN), pero en este caso es más natural que se exprese como un conjunto si se requiere empaquetamiento.

1.7.1 *Conjunto* (SET) *y vector* (ARRAY)

En las instrucciones

ident = SET OF tipobase

tipobase debe ser un tipo con un máximo de 256 elementos. Esto permite, por ejemplo, utilizar CHAR como tipobase. Sin embargo, si tipobase es un subconjunto de enteros se toma siempre como Ø..255. El tamaño en bits del conjunto es el número de elementos de tipobase. Por ejemplo, un SET OF CHAR ocupa 32 bytes (un bit por cada uno de los 256 elementos posibles).

Se admiten vectores de vectores, vectores de conjuntos, registros de conjuntos, etc.

Dos tipos ARRAY sólo se tratan como equivalentes si su definición se hace en una misma utilización de la palabra reservada **ARRAY**. En concreto, el ejemplo daría origen a dos tipos no equivalentes.

```
TYPE
        cuadroa=ARRAY[1..100] OF INTEGER;
        cuadrob=ARRAY[1..100] OF INTEGER;
```

Cualquier transferencia de los datos de cuadroa y cuadrob entre sí sería detectada como un error. En la sección 1.19 se pueden encontrar precisiones suplementarias acerca de la equivalencia. La excepción a lo que acabamos de decir la constituyen los vectores de CHAR, que se utilizan siempre para representar datos semejantes.

1.7.2 *Puntero y variable dinámica*

HP facilita la creación de variables dinámicas mediante la utilización del procedimiento normalizado NEW (véase la sección 2).

Mientras que una variable estática posee, a través de su declaración, un espacio de memoria asignado, una variable dinámica no se referencia a través de un indicador, sino de una variable puntero. La variable puntero, que es una variable estática, contiene la dirección de la variable dinámica. La variable dinámica se utiliza añadiendo el símbolo ^ al puntero.

Lo que sigue es un ejemplo de utilización de punteros (véase también el apéndice 4):

```
TYPE
    objeto=RECORD
           valor:INTEGER;
           siguiente :^objeto
           END;

     encad=^objeto;

VAR
    primero:encad;
    actual:^objeto;
```

Los punteros deben serlo de variables de tipos definidos. Sin embargo, las definiciones de tipos pueden contener punteros de sí mismos, lo que permite la creación de listas encadenadas. Es lo que ocurre justamente en el ejemplo precedente, donde el registro 'objeto' contiene su puntero 'siguiente'.

No se permiten punteros de punteros.

Dos punteros del mismo tipo son equivalentes. Así, las variables 'primero' y 'actual' del ejemplo anterior son equivalentes, pudiendo hacerse asignaciones de una a otra o comparaciones entre ambas.

Se admite la constante normalizada NIL. Cuando se asigna esta constante a una variable puntero, se considera que el puntero no contiene ninguna dirección.

1.7.3 *Registro* (RECORD)

La implementación de registros (RECORD), o sea, variables estructuradas divididas en un número fijo de campos, se hace en HP como en el Pascal normalizado, salvo por el hecho de que no se admiten partes variantes.

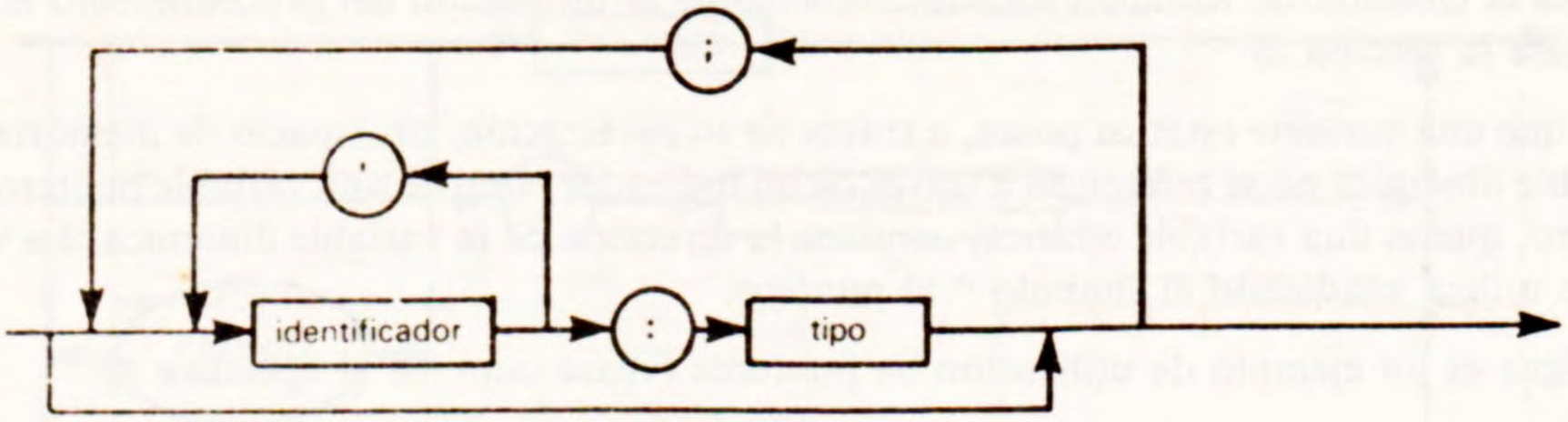
Dos variables con tipo RECORD sólo son equivalentes si su declaración se ha hecho en una misma utilización de la palabra reservada RECORD. Es semejante a lo que ya dijimos en 1.7.1.

Para acceder de una manera más compacta a los diferentes campos de un registro, se puede utilizar WITH.

Ni la declaración RECORD ni la instrucción WITH crean un contexto independiente. En consecuencia, no se debe usar el mismo identificador de campo en dos registros, ni utilizar el mismo nombre para una variable y un identificador de campo.

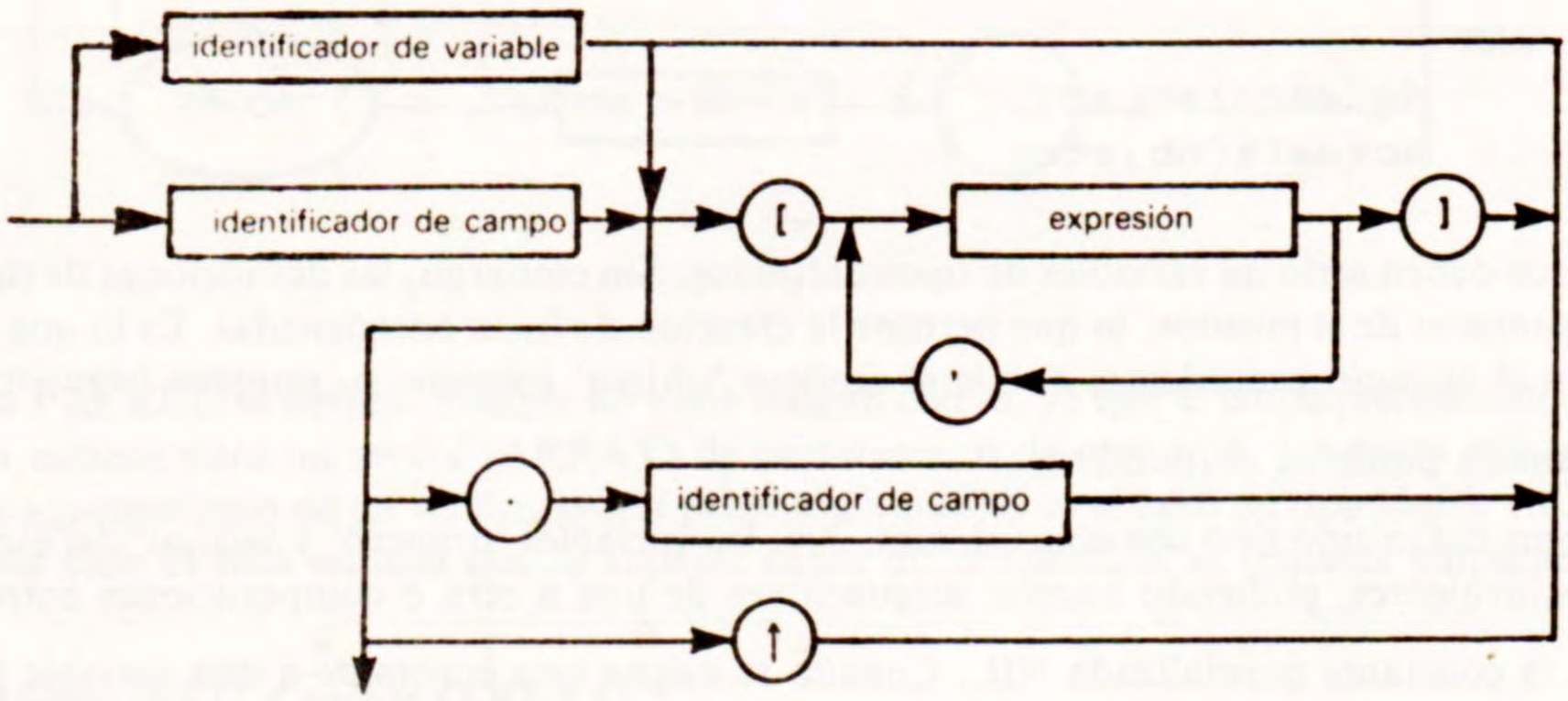
En el apéndice 4 se puede ver un ejemplo de la utilización de WITH y RECORD.

1.8 Lista de campos



Se puede ver un ejemplo en el apéndice 4.

1.9 Variable (VAR)



HP admite dos clases de variables: estáticas y dinámicas.

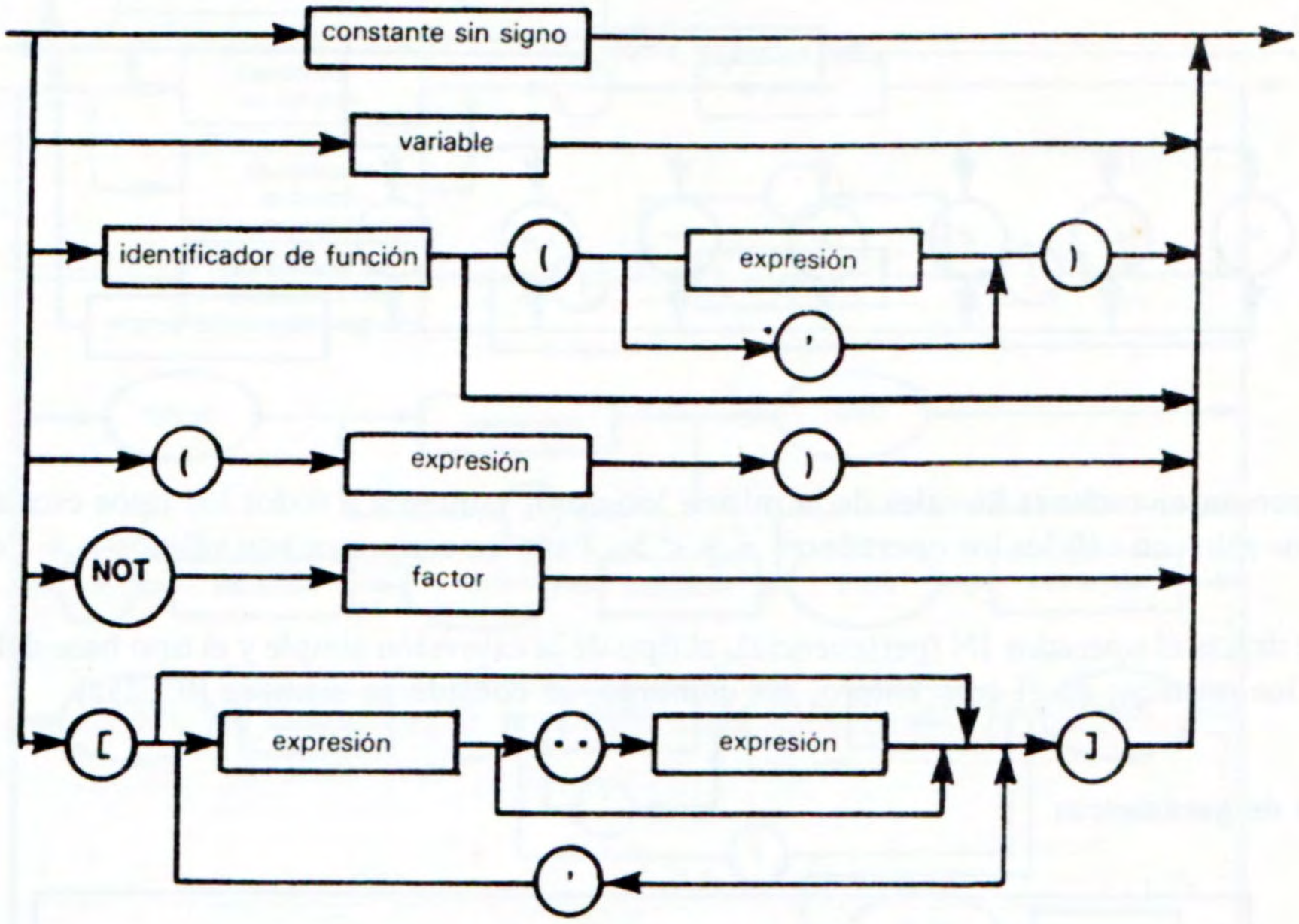
Las variables estáticas se declaran explícitamente empleando VAR y tienen reservada memoria durante toda la ejecución del bloque en el que están declaradas.

Por el contrario, las variables dinámicas se crean durante la ejecución del programa empleando NEW. No se declaran explícitamente y no se las llama mediante un identificador. Se hace referencia a ellas de manera indirecta, utilizando una variable puntero estática que contiene la dirección de la variable dinámica.

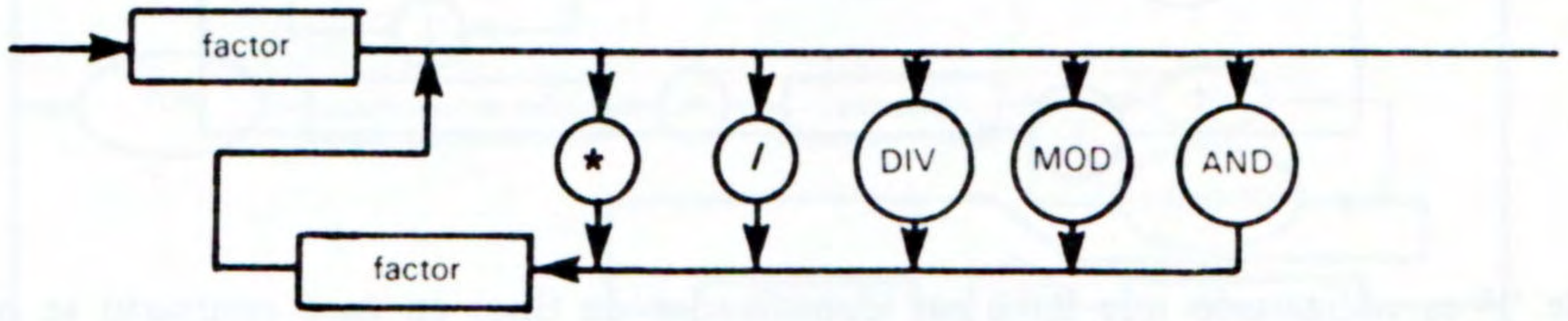
En 1.7.2 y en la sección 2 se pueden encontrar más precisiones sobre esta cuestión; el apéndice 4 contiene ejemplos.

La forma de especificar un elemento de un vector multidimensional no es forzosamente la misma que se ha empleado en la declaración. Por ejemplo, si la variable 'mat' se ha declarado como 'ARRAY[1..1Ø] OF ARRAY[1..1Ø]', se puede llamar al elemento (1,1) de la matriz como 'mat[1][1]' o 'mat[1,1]'.

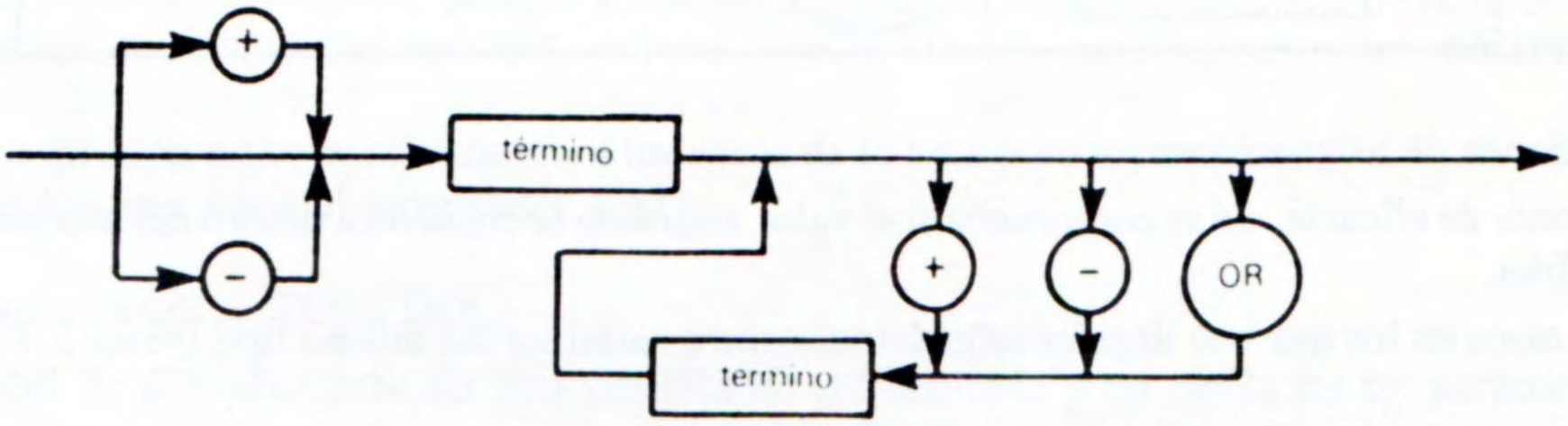
1.10 Factor



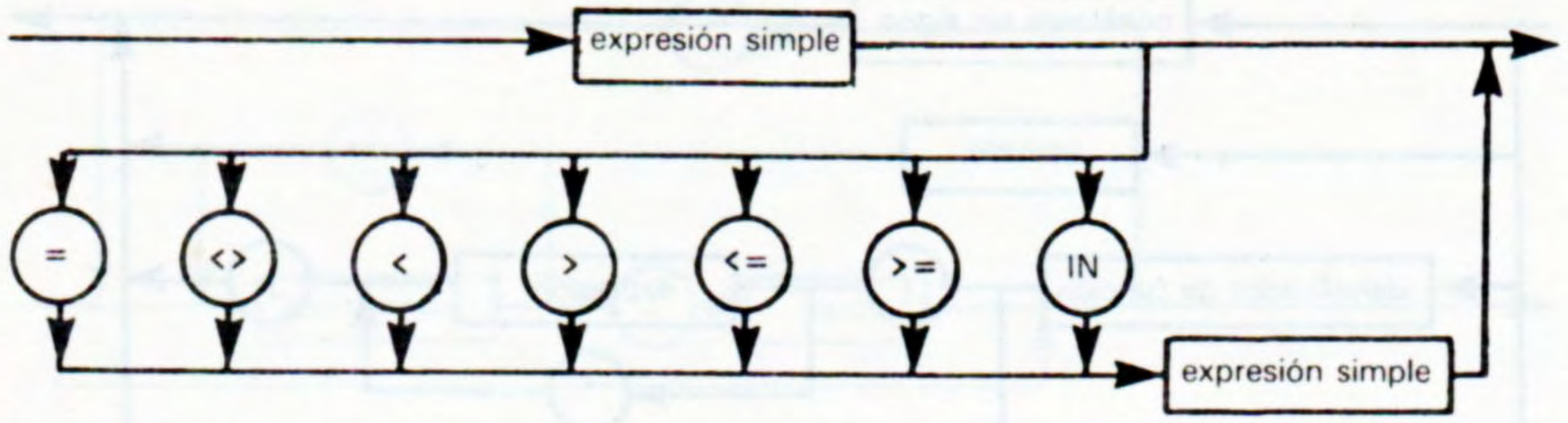
1.11 Término



1.12 Expresión simple



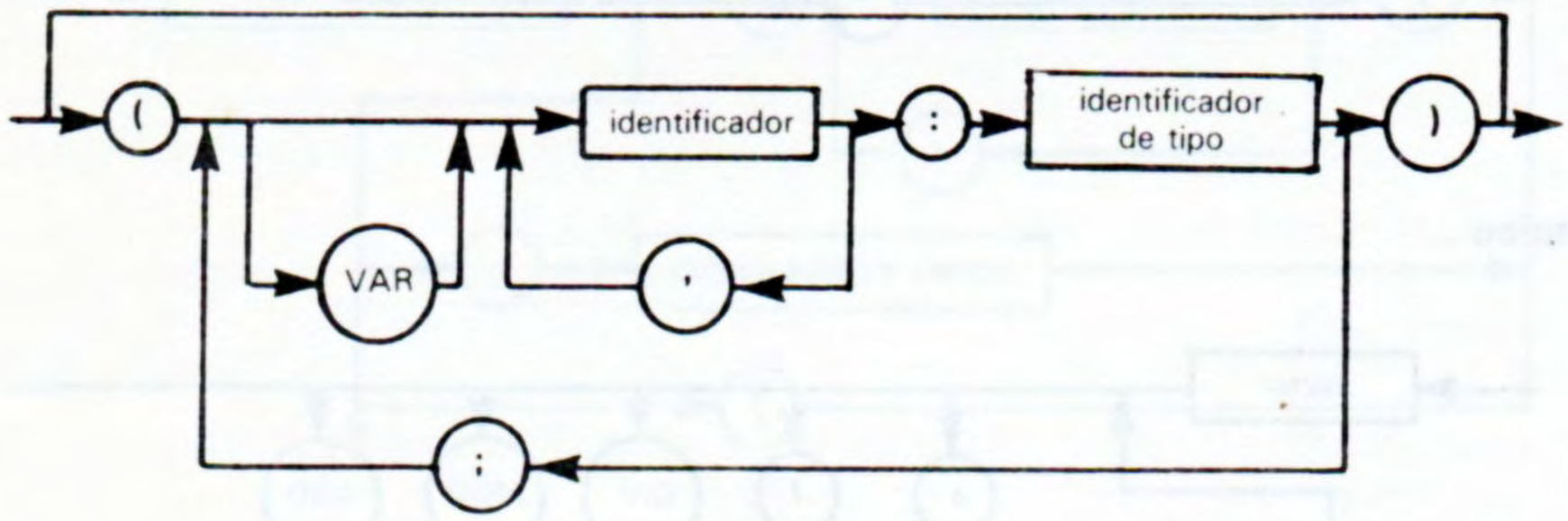
1.13 Expresión



Se puede comparar cadenas literales de la misma longitud, punteros y todos los tipos escalares. Para los punteros sólo son válidos los operadores = y <>. Para los conjuntos son válidos >=, <=, <> y =.

Cuando se utiliza el operador IN (pertenencia), el tipo de la expresión simple y el tipo base del conjunto deben ser los mismos; en el caso entero, sin embargo, se consideran siempre [Ø..255].

1.14 Lista de parámetros



Después de ':' es obligatorio que haya un identificador de tipo; en caso contrario se produce el '*ERROR*14'.

Los parámetros pueden ser tanto variables como valores; los procedimientos (PROCEDURES) y las funciones (FUNCTION) no son parámetros válidos en HP.

1.15 Instrucción

—Instrucciones de asignación:

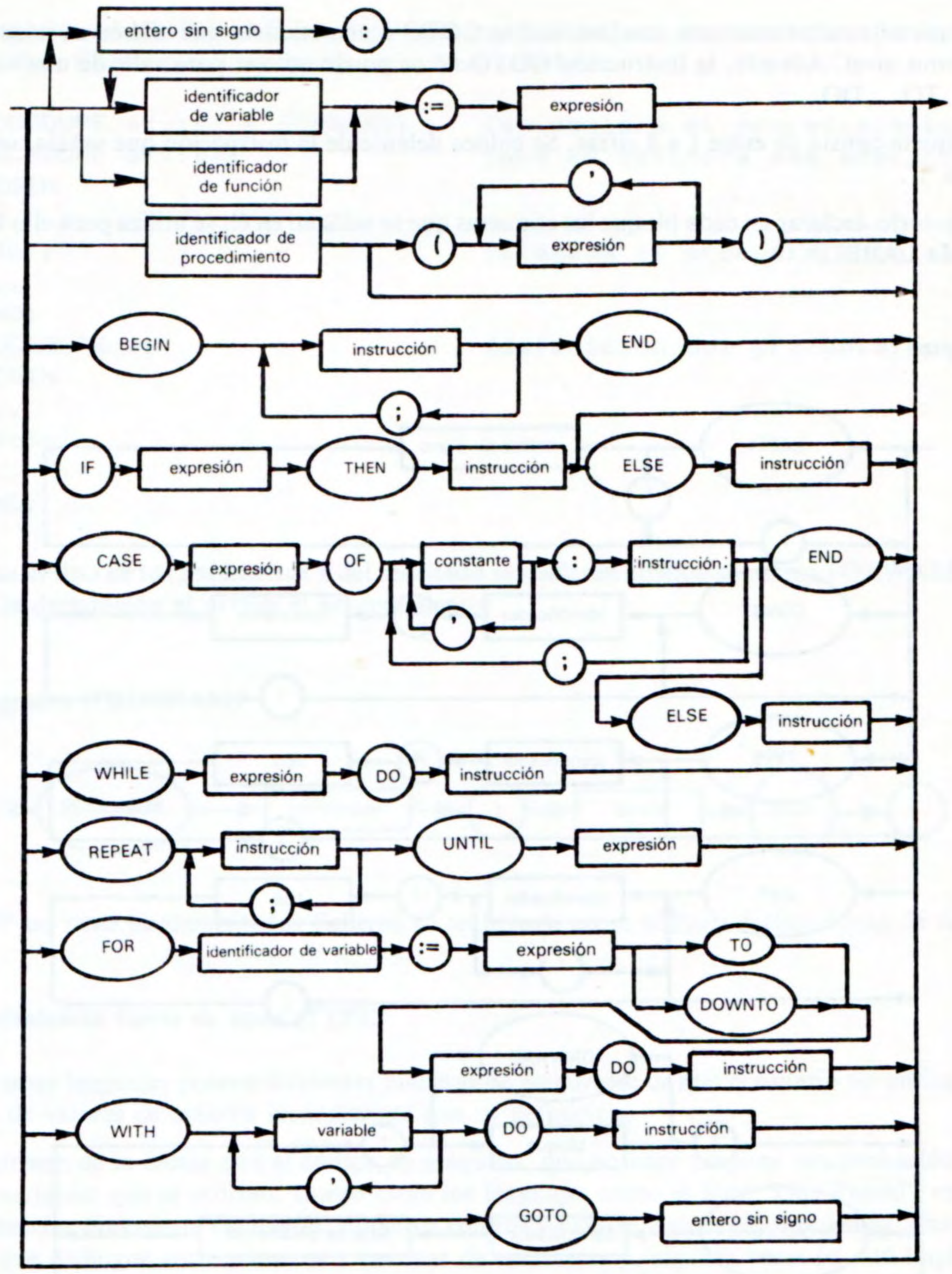
Por razones de eficacia, no se comprueba si el valor asignado se encuentra dentro del margen de valores posibles.

Existen casos en los que son ilegales asignaciones entre variables del mismo tipo (véase 1.7.1 y 1.19).

—Instrucciones de elección múltiple (CASE):

No se permite una lista nula de casos posibles. La instrucción 'CASE OF END;' genera el '*ERROR*13'.

La parte ELSE se ejecuta cuando la expresión que actúa de selector no se encuentra en ninguno de los casos de la lista.



Cuando la expresión no se encuentra entre los casos de la lista y no existe parte ELSE, el control pasa a la instrucción que sigue al terminador END.

—Instrucciones FOR...TO...DO:

La variable de contador debe ser una variable no estructurada y no puede ser un parámetro. Esto está a medio camino entre la normalización Jensen/Wirth y el proyecto ISO.

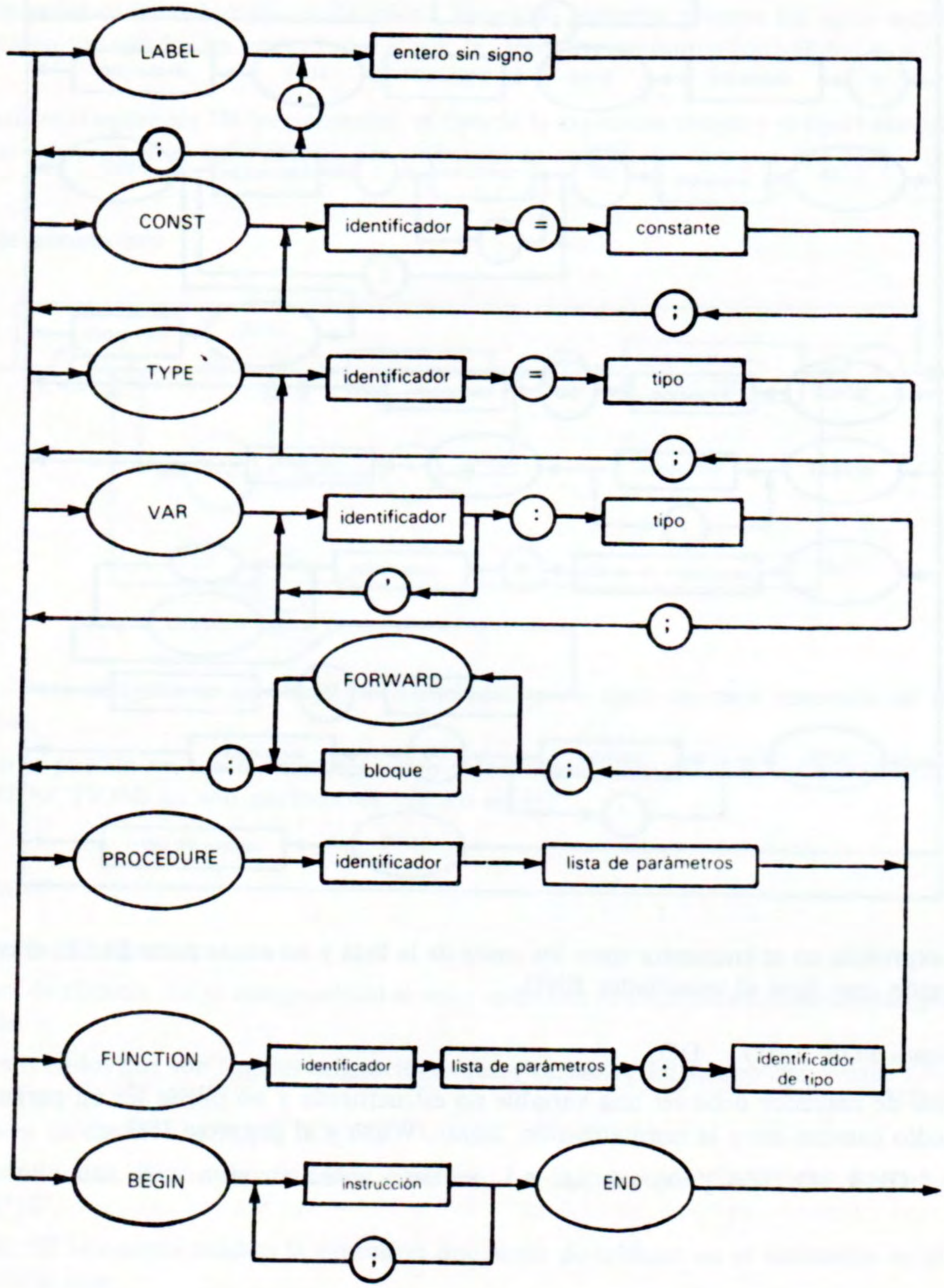
—Instrucciones GOTO:

Sólo se permite saltar mediante una instrucción GOTO a una etiqueta que esté en su mismo bloque y al mismo nivel. Además, la instrucción GOTO no se puede utilizar para salir de una instrucción FOR...TO...DO.

Una etiqueta consta de entre 1 a 4 cifras. Se coloca delante de la instrucción que señala, seguida del símbolo ':'.

Es obligatorio declarar en cada bloque las etiquetas que se utilizan en él; se utiliza para ello la palabra reservada LABEL.

1.16 Bloque



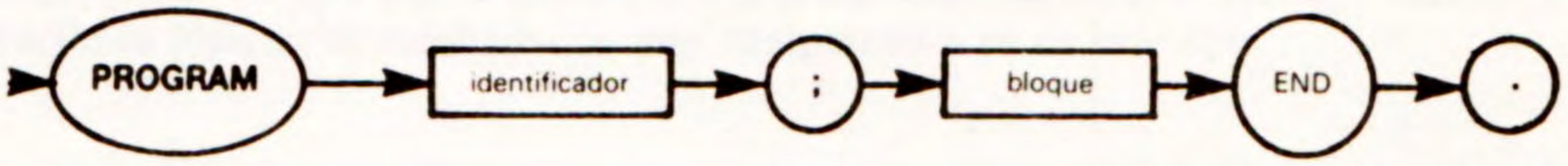
1.17 Referencia adelante (FORWARD)

La palabra reservada FORWARD permite llamar una función o un procedimiento antes de que hayan sido definidos. El ejemplo siguiente muestra cómo hacerlo:

```
PROCEDURE a(y:t) ; FORWARD;      {se declara el procedimiento a}
PROCEDURE b(x:t);                {que se definira mas adelante}
 BEGIN
 ...
 a(p);                           {llamada al procedimiento a}
 ...
 END;
PROCEDURE a;                     {definicion del procedimiento a}
 BEGIN
 ...
 b(q);
 ...
 END;
```

Nótese que el tipo de los parámetros y del resultado se declaran cuando se utiliza FORWARD, sin que se repita la declaración al definir el procedimiento.

1.18 Programa (PROGRAM)



Como HP no tiene implementados ficheros (o archivos), no se utilizan declaraciones de fichero.

1.19 Equivalencia fuerte de tipos (TYPE)

Los diferentes lenguajes poseen diferentes maneras de asegurarse de que el usuario no utiliza los diversos tipos de valores de manera inconsistente con su definición.

En un extremo de la escala está el código de máquina, que no hace ninguna comprobación sobre los tipos de variables que se utilizan. Luego están los lenguajes como el Byte 'Tiny Pascal', en el que se pueden mezclar libremente caracteres, números enteros y valores lógicos sin problemas. Después viene BASIC, que distingue entre números y cadenas de caracteres y, algunas veces (si está implementado el símbolo %), entre números enteros y reales. Luego va PASCAL, que permite diferentes tipos de valores creados por el usuario. Al final de la escala (por el momento) está un lenguaje como ADA, que permite definir tipos de números diferentes e incompatibles.

Las distintas implementaciones de Pascal utilizan básicamente dos sistemas para reforzar la separación de los tipos de datos: equivalencia estructural y equivalencia de nombre. HP utiliza la equivalencia de nombre en el caso de registros (RECORD) y vectores (ARRAY). Ya hablamos de este tema en 1.7.1; vamos ahora a clarificar este concepto con un ejemplo. Si se definen dos variables mediante

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

puede parecer que serán correctas asignaciones tales como 'A:=B;'. Sin embargo, HP considera esto incorrecto y produce el mensaje de '*ERROR*1Ø'. Si se desea que A y B sean realmente del mismo tipo (sean equivalentes) para que puedan hacerse asignaciones entre ambas variables, se las debe definir mediante

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

Aunque este sistema de la equivalencia de nombre puede parecer un poco complicado, tiene la ventaja de disminuir los errores de programación, ya que exige una verdadera intencionalidad del programador para que se produzca la equivalencia.

SECCIÓN 2. IDENTIFICADORES PREDEFINIDOS

2.1 Constantes (CONST)

MAXINT

Es el mayor entero aceptable, o sea, 32767.

TRUE, FALSE

Constantes lógicas (verdadero y falso).

2.2 Tipos (TYPE)

INTEGER, REAL, CHAR, BOOLEAN

Los tipos entero y real han sido comentados en la sección 1.3. El tipo carácter (CHAR) comprende los 256 caracteres ASCII. El tipo lógico (BOOLEAN) comprende los valores TRUE y FALSE y se utiliza en las operaciones lógicas; el resultado de una comparación es de este tipo.

2.3 Procedimientos (PROCEDURE) y funciones (FUNCTION)

2.3.1 *Procedimientos para entradas y salidas*

procedimiento WRITE

Se emplea para emitir datos por la pantalla o la impresora.

La forma de la correspondiente instrucción es

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

que es equivalente a

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

Los parámetros P1, P2, . . . , Pn pueden ser de una de las formas siguientes:

e e:m e:m:n e:m:H

donde e, m y n son expresiones y H es la letra hache. Veamos lo que ocurre con WRITE (P) en cada uno de los casos posibles.

1) Si e es de tipo entero se pueden usar los formatos decimales 'e' o 'e:m'.

El valor del número entero e se transforma en una cadena literal. Si m no figura, entonces se escribe la cadena con un blanco de separación. Si está presente m, entonces la cadena se completa con blancos

a la izquierda hasta que tenga m caracteres y se escribe la cadena. Si m caracteres no bastan para representar el número, se ignora m. Si m coincide con la longitud del número, sin el blanco de separación, entonces el blanco no se escribe.

2) Si e es de tipo entero se puede usar el formato hexadecimal 'e:m:H'.

Se escribe el valor del número e en hexadecimal. Si m = 1 o m = 2, lo que se escribe es el valor de 'e MOD 16^m' con m cifras. Si m = 3,4,5,. . ., entonces se escribe el valor de e con 4 cifras (con algún Ø a la izquierda si es necesario) y ocupando 4 lugares si m = 3 o m = 4, o m lugares si m > 4. Así, por ejemplo

```
WRITE(1Ø25:m:H);

m=1    la salida es: 1
m=2    la salida es: Ø1
m=3    la salida es: Ø4Ø1
m=4    la salida es: Ø4Ø1
m=5    la salida es: _Ø4Ø1
```

3) Si e es de tipo real se pueden usar los formatos 'e', 'e:m' o 'e:m:n'.

Con el formato 'e:m' el número se escribe en notación científica, con mantisa y exponente. Si el número es negativo, irá precedido del signo menos; si no, se pone un blanco en el lugar del signo. Las cifras decimales pueden oscilar entre 1 (mínimo) y 5. El exponente lleva siempre signo (más o menos) y dos cifras. Todo esto hace un mínimo de 8 caracteres y un máximo de 12. Es m el que determina el número de caracteres entre 8 y 12. Si m < 8, se ocupan los 12 caracteres. Si m está entre 8 y 12, se ocupan m caracteres poniendo los decimales que sean necesarios. Si m > 12, el número tendrá 12 caracteres, pero irá precedido de los blancos que sean necesarios. Por ejemplo,

```
WRITE(-1.23E 1Ø:m);

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

Con el formato 'e:m:n' el número e se escribe en notación de punto fijo, ocupando m posiciones y con n cifras tras el punto decimal. Si m es excesivo, se escriben blancos a la izquierda. Si n es Ø, se escribe e como un entero. Si e es excesivamente grande para el formato, se ignora n y se emplea la notación científica en la forma descrita antes. Por ejemplo,

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

4) Si e es de tipo carácter o cadena literal, se pueden usar los formatos 'e' o 'e:m'.

Se escribirá el literal e con una longitud mínima igual a la del literal (1 si es un carácter). Si m es excesivamente grande, se escriben blancos a la izquierda hasta completar m.

Cuando e es un carácter de control, WRITE(e) realiza la acción pertinente. Así, CHR(8) (⟨CTRL/H⟩) retrocede borrando un espacio, CHR(12) (⟨CTRL/L⟩) borra la pantalla o salta a la página siguiente en la impresora, CHR(13) (⟨CTRL/M⟩) hace que se salte al comienzo de la línea siguiente, CHR(16) (⟨CTRL/P⟩) cambia la salida normal de pantalla a impresora o viceversa.

5) Si e es de tipo lógico, se pueden usar los formatos 'e' o 'e:m'.

La salida es 'TRUE' o 'FALSE' según el caso (no ocupan menos de 4 o 5 caracteres).

procedimiento WRITELN

Es como WRITE, pero pasa al comienzo de la línea siguiente tras realizar la impresión. Por ejemplo,

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

equivale a

```
BEGIN WRITE(P1,P2,...Pn);WRITELN END;
```

procedimiento PAGE

Equivale a WRITE(CRH(12)); y tiene por efecto borrar la pantalla o saltar a la página siguiente en la impresora.

procedimiento READ

Se utiliza para introducir datos con el teclado.

Esto se hace a través de un tampón que recibe, cuando el programa lo ordena, los caracteres del teclado. De entrada, el tampón no tiene más que un carácter de fin de línea. La lectura del contenido del tampón se realiza a través de una ventana de texto que permite ver un carácter cada vez. Cuando aparece un carácter de fin de línea en la ventana de texto, se da la orden de aceptar una nueva línea del teclado. Los caracteres de control que se pulsen (véase la sección Ø.2) son reconocidos como tales.

La forma de la correspondiente instrucción es

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

que es equivalente a

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

donde V1, V2, . . ., Vn pueden ser de tipo carácter, cadena literal, entero o real.

La instrucción 'READ(V);' provoca efecto distinto según el tipo de V. Trataremos los cuatro casos posibles.

1) *V de tipo carácter*

En este caso, READ(V) lee un carácter del tampón de entrada del teclado y lo asigna a V. Si el carácter es el CHR(13) de fin de línea, la función EOLN toma el valor TRUE y se da la orden de aceptar una nueva línea del teclado, o sea, la siguiente operación de lectura comenzará con la ventana de texto al comienzo de una nueva línea.

Nota importante. Al comienzo del programa, la función EOLN toma el valor TRUE. Si la primera instrucción READ es de tipo carácter, el carácter que se lee es CHR(13) y se produce un cambio de línea. La siguiente operación de lectura comenzará entonces por el primer carácter de la nueva línea. Se puede utilizar el procedimiento READLN para superar el inconveniente que supone el que la primera línea que se lee esté en blanco.

2) *V de tipo cadena literal*

En este caso se lee una serie de caracteres hasta completar la longitud de la cadena o hasta que EOLN tome el valor TRUE. Si lo que sucede es esto último y la cadena no queda completa, se rellenan con CHR(Ø) (el carácter nulo) los lugares últimos de la cadena; esto permite calcular la longitud de la cadena que se lee. También hay que tener en cuenta lo que sucede en la primera operación de lectura y que hemos explicado en la nota anterior.

3) *V de tipo entero*

En este caso se lee una serie de caracteres que representen un entero. Se pasan por alto los blancos y caracteres de fin de línea previos. Por esta razón no existen problemas con la primera operación de lectura cuando lo es de un entero.

Si el entero que se lee es mayor que MAXINT (32767), se produce el mensaje de error 'Number too large' en el momento de la ejecución y ésta se detiene. Si el primer carácter que se lee (fuera de los blancos y fin de línea) no es un signo '+' o '−' o un digito, se produce el mensaje de error 'Number expected' en el momento de la ejecución y ésta se detiene.

4) *V de tipo real*

En este caso se lee una serie de caracteres que representen un número real. Se pasan por alto los blancos y caracteres de fin de línea previos. Por lo tanto, tampoco existen en este caso problemas con la primera operación de lectura.

Durante la ejecución se pueden producir los siguientes mensajes de error, que detienen además la ejecución: 'Number expected' cuando el primer símbolo no es ni el signo ni un dígito, o cuando el punto decimal no lleva después ningún dígito; 'Overflow' cuando el número es demasiado grande o demasiado pequeño (véase la sección 1.3); 'Exponent expected' cuando la letra 'E' no va seguida ni de un signo ni de un dígito.

procedimiento READLN

Es como READ, pero hace que tras la lectura se pase a una nueva línea. Después de la ejecución de READLN la función EOLN toma el valor FALSE salvo que la siguiente línea esté en blanco. Por ejemplo, la instrucción

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

equivale a

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

Se puede utilizar READLN para evitar los problemas que se crean en la primera lectura del programa, cuando lo es de un carácter o cadena literal. El procedimiento READLN hace que se salte la primera línea (la línea que de entrada está en blanco).

Como en el caso de READ, se pueden utilizar los caracteres que proporcionan facilidades de edición. Observe, sin embargo, que sólo pasan al tampón del teclado los caracteres que realmente se pulsan, ignorándose los caracteres que pueda haber sobre la pantalla.

2.3.2 *Funciones para entradas*

función EOLN

Es una función lógica (BOOLEAN) que toma el valor TRUE cuando el siguiente carácter que se va a leer es un fin de línea (CHAR(13)), y el valor FALSE en cualquier otro caso.

función INCH

Esta función entrega como valor un carácter. Si se ha pulsado una tecla, entrega el carácter correspondiente a la tecla; si no, entrega el carácter CHR(Ø).

La función es independiente de las comprobaciones de teclado que corresponden a la opción '$C' (véase la sección 3.2); así pues, no es necesario especificar la opción '$C–'.

2.3.3 *Funciones de transferencia entre tipos*

función TRUNC(X)

X debe ser de tipo real o entero. La función entrega como valor el mayor entero menor o igual que X cuando X es positivo, o el menor entero mayor o igual que X cuando X es negativo. Es decir, suprime en cualquier caso la parte decimal del número. Por ejemplo, TRUNC(–1.5) da –1 y TRUNC(1.9) da 1.

función ROUND(X)

X debe ser de tipo real o entero. La función entrega como valor el entero más próximo a X (con las reglas normales de redondeo). Por ejemplo, ROUND(–6.5) da –6, ROUND(–6.51) da –7, ROUND(11.7) da 12 y ROUND(23.5) da 24.

función ENTIER(X)

X debe ser de tipo real o entero. La función entrega el mayor entero menor o igual que X. Por ejemplo, ENTIER(–6.5) da –7 y ENTIER (11.7) da 11.

Esta función de HP no es una función de Pascal normalizado; equivale a la función INT de BASIC.

función ORD(X)

X puede ser de cualquier tipo escalar, excepto real. La función entrega un entero que es el número de orden que ocupa el valor que tenga X en el conjunto que define el tipo de X. Si X es de tipo entero entonces ORD(X)=X, por lo que no se la utiliza en ese caso. Por ejemplo, ORD('a') da 97 y ORD ('[') da 91.

función CHR(X)

X debe ser de tipo entero. La función entrega el carácter cuyo código ASCII es el valor de X. Por ejemplo, CHR(97) da 'a' y CHR(91) da '['.

2.3.4 *Funciones matemáticas*

En todas ellas el parámetro X debe ser de tipo real o entero.

función ABS(X)

Entrega el valor absoluto de X; por ejemplo, ABS(–4.5) da 4.5. El resultado es del mismo tipo que X.

función SQR(X)

Entrega X*X, o sea, el cuadrado de X. El resultado es del mismo tipo que X.

función SQRT(X)

Entrega la raíz cuadrada de X cuando X es positivo o Ø. Genera el mensaje 'Maths Call Error' cuando X es negativo. El resultado es siempre de tipo real.

función FRAC(X)

Entrega la parte fraccionaria de X, es decir, FRAC(X) = X – ENTIER(X). Por ejemplo, FRAC(1.5) da Ø.5 y FRAC(– 12.56) da Ø.44.

función SIN(X)

Entrega el seno de X (que se interpreta en radianes). El resultado es siempre de tipo real.

función COS(X)

Entrega el coseno de X (que se interpreta en radianes). El resultado es siempre de tipo real.

función TAN(X)

Entrega la tangente de X (que se interpreta en radianes). El resultado es siempre de tipo real.

función ARCTAN(X)

Entrega el ángulo (en radianes) cuya tangente es X. El resultado es siempre de tipo real.

función EXP(X)

Entrega el valor e^X, con e = 2.71828. El resultado es siempre de tipo real.

función LN(X)

Entrega el logaritmo natural (neperiano) de X cuando X es positivo. Genera el mensaje 'Maths Call Error' cuando X es negativo o Ø. El resultado es siempre de tipo real.

2.3.5 *Otros procedimientos predefinidos*

procedimiento NEW(P)

Reserva espacio para una variable dinámica cuyo puntero es P. Tras la ejecución de NEW(P), P contiene la dirección de la nueva variable dinámica. Para acceder a ella se utiliza P^. En el apéndice 4 se puede ver un ejemplo de utilización.

El espacio reservado para la variable dinámica se puede reutilizar empleando los procedimientos MARK y RELEASE.

procedimiento MARK(Pm)

La variable dinámica se acumula en la memoria en forma de pila (se suele hablar de "heap", montículo). El procedimiento MARK(Pm) fija en el puntero Pm la dirección de la parte superior de la pila de la variable a la que apunta. La memoria ocupada por la variable dinámica desde el momento de la utilización de MARK se puede liberar en un momento dado mediante el procedimiento RELEASE. Véase en el apéndice 4 un ejemplo de utilización.

El puntero Pm de la variable dinámica debe ser diferente del utilizado por el procedimiento NEW.

procedimiento RELEASE(Pm)

Libera el espacio utilizado por la variable dinámica desde el momento de la ejecución de MARK(Pm), destruyendo así las variables dinámicas creadas a partir de la instrucción MARK. En consecuencia *se lo debe utilizar con sumo cuidado*. Véase en el apéndice 4 un ejemplo de utilización.

procedimiento INLINE(N1,N2,N3,...)

Permite insertar código de máquina en medio del programa Pascal. Los valores N1 MOD 256, N2 MOD 256, N3 MOD 256, etc. se insertarán en el código objeto en la dirección que el contador de posición del compilador posea en ese momento. Los parámetros N1, N2, N3, etc. pueden ser constantes enteras cualesquiera. Véase en el apéndice 4 un ejemplo de utilización.

procedimiento USER(N)

El argumento N debe ser entero. El procedimiento USER(N) produce una llamada a la dirección de memoria dada por N. La rutina a la que se llama debe terminar con la instrucción RET (#C9) y preservar el registro IX.

Recuérdese que, si un programa Pascal se está ejecutando en un sistema con 64K, estará presente la RAM por debajo de #8000, lo que hará que no se pueda utilizar la ROM de BASIC.

Puesto que representa una dirección de memoria, es conveniente dar N en forma hexadecimal. Si se utiliza la forma decimal, hay que recordar que HP trata los enteros en la forma de complemento a 2 (véase el apéndice 3). En consecuencia, las direcciones mayores que 32767 (#7FFF) deben ser introducidas como números negativos, si se hace en forma decimal. Por ejemplo, 'USER(-16384);' llamará a la posición #C000.

procedimiento HALT

Detiene la ejecución del programa y genera el mensaje 'Halt at PC=XXXX', donde XXXX es la dirección de memoria en la que se ha detenido el programa (en hexadecimal). Este procedimiento se puede utilizar, con la ayuda de un listado de compilación, para seguir la trayectoria de un programa en caso de posibles bifurcaciones: se lo utiliza, pues, en la fase de depuración de los programas.

procedimiento POKE(N,V)

Almacena la expresión V en la memoria del ordenador a partir de la dirección de memoria N. El argumento N debe ser de tipo entero y el V de cualquier tipo salvo SET. En lo que respecta a la forma en que debe darse N, se puede decir lo mismo que explicábamos para el procedimiento USER. Por ejemplo, POKE(#6000, 'A') colocará #41 en la posición #6000 y POKE(-16384,3.6E3) colocará #00 #0B #80 #70 en la posición #C000.

procedimiento TOUT(nombre,comienzo,tamaño)

Se lo utiliza para almacenar en cinta los valores de las variables. Los parámetros comienzo y tamaño son de tipo entero; se almacenará en cinta el número de bytes de memoria dado por 'tamaño' tomados a partir de la posición 'comienzo'. El almacenamiento se hace en un fichero cuyo nombre viene dado por el primer parámetro; este parámetro debe ser de tipo 'ARRAY[1..8] OF CHAR'.

Por ejemplo, se puede grabar la variable V en un fichero de cinta con nombre 'VAR ' utilizando la instrucción

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

en la que aparecen las funciones ADDR y SIZE que veremos más adelante. También se puede ver un ejemplo en el apéndice 4.

La utilización de direcciones de memoria (en lugar de utilizar directamente las variables) proporciona mayores recursos al programador. Así, si un sistema posee un área de memoria para la pantalla, se puede grabar directamente todo el contenido de la pantalla.

procedimiento TIN(nombre,comienzo)

Se utiliza para cargar de la cinta los valores de las variables que han sido grabados con el procedimiento TOUT. El tipo y significado de los parámetros es el mismo que en TOUT; el parámetro 'tamaño' no aparece, puesto que se utilizará el mismo tamaño que se dio al grabarlo con TOUT.

Por ejemplo, cargaremos la variable almacenada en el ejemplo precedente mediante la instrucción

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

También se puede ver un ejemplo en el apéndice 4.

procedimiento OUT(P,C)

Manda el parámetro carácter C a la puerta del Z8Ø dada por el parámetro entero P. El valor de P se carga en el registro BC, el carácter C se carga en el acumulador y se ejecuta la instrucción 'OUT (C),A' del Z8Ø. Por ejemplo, OUT(1,'a') se utiliza para sacar el carácter 'a' por la puerta 1 del Z8Ø.

Nótese que lo mismo se puede hacer utilizando el procedimiento INLINE.

procedimiento RANSEED(X,Y,Z)

Hace que los enteros X, Y y Z se utilicen como semilla para construir un número aleatorio mediante la función RANDOM. Los parámetros deben estar entre 1 y 3ØØØØ.

Si se quiere utilizar los mismos números aleatorios en todas las ejecuciones del programa, se debe usar este procedimiento.

2.3.6 *Otras funciones predefinidas*

función RANDOM

Entrega un número real pseudoaleatorio comprendido entre Ø.Ø y 1.Ø. Utiliza un algoritmo basado en el de B.A. Wichmann y I.D. Hill (NPL Report DITC 6/82). Mediante el procedimiento RANSEED se pueden utilizar como semilla los números que se desee.

función SUCC(X)

El argumento X puede ser de cualquier tipo escalar, pero no REAL. La función entrega el sucesor de X en el conjunto que define el tipo de X. Por ejemplo, SUCC('A') da 'B' y SUCC(5) da 6.

función PRED(X)

El argumento X puede ser de cualquier tipo escalar excepto REAL. La función entrega el predecesor de X en el conjunto que define el tipo de X. Por ejemplo, PRED ('j') da 'i' y PRED(TRUE) da FALSE.

función ODD(X)

X debe ser de tipo entero. La función entrega el valor lógico TRUE si X es impar (''odd'') y FALSE si X es par.

función ADDR(V)

Utiliza como parámetro una variable de cualquier tipo. Entrega un valor entero que es la dirección de memoria de la variable. En el apéndice 3 se puede ver información acerca del modo de almacenamiento de las variables durante la ejecución. El apéndice 4 muestra un ejemplo de utilización de ADDR.

función SIZE(V)

Utiliza como parámetro una variable y entrega un valor entero que es la cantidad de bytes que ocupa en memoria. Puede ser utilizada en combinación con ADDR.

función PEEK(N,T)

El primer argumento de la función es un número entero que se interpreta como una dirección de memoria (con las mismas consideraciones que hicimos para el procedimiento USER). El segundo argumento es un tipo; el valor de la función será de este tipo.

PEEK entrega el valor del tipo T almacenado en la posición de memoria N; es la función opuesta de POKE. La acción concreta de PEEK y de POKE y el número de bytes afectados depende de la representación de los diversos tipos (véase el apéndice 3).

Por ejemplo, si a partir de la dirección #5000 se encuentran los valores #50 #61 #73 #63 #61 #6C, entonces

```
WRITE(PEEK(#5000,ARRAY[1..6] OF CHAR))   da 'Pascal'
WRITE(PEEK(#5000,CHAR))                  da 'P'
WRITE(PEEK(#5000,INTEGER))               da 24912
WRITE(PEEK(#5000,REAL))                  da 2.46227E+29
```

función INP(P)

Permite acceder a una puerta del Z80. El valor del parámetro entero P se carga en el registro BC, se ejecuta la instrucción 'IN A,(C)' del Z80 y el carácter que haya en el acumulador será el valor que entregue INP.

Nótese que lo mismo se puede hacer utilizando el procedimiento INLINE.

SECCIÓN 3. COMENTARIOS Y OPCIONES DE COMPILACIÓN

3.1 Comentarios

Un comentario puede aparecer situado entre palabras reservadas, números, identificadores o símbolos especiales.

Un comentario comienza por el símbolo '{' o '(*' y termina por el correspondiente símbolo de cerrar '}' o '*)'. Todos los caracteres que se encuentren entre los símbolos de comienzo y fin son ignorados por el compilador, excepto cuando aparece el símbolo '$'. Si encuentra '$' en un comentario, el compilador interpreta los símbolos que siguen como opciones de compilación.

3.2 Opciones de compilación

Las opciones de compilación, que tienen por efecto modificar la manera en que se efectúa la compilación, se incluyen dentro de los símbolos delimitadores de los comentarios. La primera opción en la lista de opciones debe ir precedida del símbolo '$'.

Por ejemplo, '(*$C-,A-*)' activaría las opciones de compilación 'C-' y 'A-'.

Algunas opciones tienen interés mientras se depura el programa. Cuando una sección del programa ha sido depurada, se las puede desactivar para dar mayor velocidad al programa, activándolas a su vez en secciones aún no comprobadas.

opción L acción: control del listado de compilación
por defecto: L+

Controla el listado de compilación que contiene el programa fuente y la dirección del código objeto correspondiente a cada línea.

Con L+ se obtiene un listado completo.

Con L- se listan solamente las líneas en que aparecen errores.

opción O acción: control de sobrepasamiento en suma y resta de enteros
por defecto: O+

Con O+ se controla el sobrepasamiento ("overflow") en la suma y la resta de tipos enteros.

Con O- no se controla en esos casos.

El sobrepasamiento se controla en cualquier caso para todas las operaciones de tipo real y para la multiplicación y división entera.

opción C acción: controla si se pulsa ⟨CTRL/STOP⟩
por defecto: C+

Con C+ el programa controla durante la ejecución si se pulsa ⟨CTRL/STOP⟩ y, en caso afirmativo, la ejecución se detiene, aparece el mensaje 'Halt' (consulte el procedimiento HALT en la sección 2.3.5) y el control vuelve al editor. El control del teclado y la eventual detención se realiza al comienzo de todos los bucles, procedimientos y funciones. Se utiliza durante el proceso de depuración del programa, pero se lo debe desactivar si se desea una ejecución rápida.

Con C− no se efectúa este control del teclado.

opción S acción: controla el posible desbordamiento de la pila
por defecto: S+

Con S+ se controla al comienzo de cada llamada a un procedimiento o a una función la *posibilidad* de que exista desbordamiento de la pila. Si se prevé que la pila pueda invadir la memoria utilizada por una variable dinámica u ocupada por el programa, se detiene la ejecución y aparece el mensaje 'Out of RAM at PC = XXXX'. Este control no es infalible; si un procedimiento utiliza en sí mismo una gran cantidad de espacio para la pila, el programa puede fallar sin que el control realizado lo haya previsto; por el contrario, una llamada recursiva a una función puede hacer que el control detecte como probable un desbordamiento y detenga el programa, aunque el temor sea injustificado si la función hace poco uso de la pila.

Con S− no se controla el posible desbordamiento de la pila.

opción A acción: controla que los índices de los vectores estén en el intervalo correcto
por defecto: A+

Si se utiliza A+ y un índice de un vector toma valores fuera del intervalo dado en la definición, se detiene la ejecución y aparece el mensaje 'Index too high'o 'Index too low' que corresponda.

Con A− no se controla el valor de los índices.

opción I acción: asegura la correcta comparación de dos enteros
por defecto: I−

Debido al uso del complemento a 2 para representar los enteros en 16 bits, el resultado de una comparación >, <, >= o (= entre enteros puede ser incorrecto cuando éstos difieren en más de MAXINT (32767).

La utilización de I+ garantiza, aun en este caso, la correcta comparación de dos enteros.

Con I− no se puede asegurar la validez del resultado en el caso expuesto.

Lo mismo puede ocurrir cuando se comparan dos reales que difieren en más de 3.4E38 aproximadamente, pero I+ no pone remedio a esto.

opción P acción: envía el listado a la pantalla o a la impresora
por defecto: se utiliza la pantalla

Cambia el periférico por el que se emite el listado, de pantalla a impresora o viceversa. De entrada se utiliza la pantalla. Nótese que esta opción no lleva la especificación '+' o '−', ya que es en sí misma un conmutador.

opción F nomfiche acción: incluye un programa Pascal

En esta opción, la letra F debe ir seguida de un blanco y de un nombre de fichero de hasta 8 caracteres; el nombre debe completarse con blancos si tiene menos de 8 caracteres.

La presencia de esta opción en una línea hace que se incluya al final de la línea el texto de un fichero fuente Pascal previamente grabado en cinta con el nombre que figura tras la F. El fichero que se desee incluir ha debido ser grabado mediante el comando 'P' del editor.

Por ejemplo, '{$F MATRIZ incluye el procedimiento MATRIZ}' sirve para incluir el fichero de cinta que tiene el nombre de 'MATRIZ'.

Puede ser conveniente utilizar en este caso la opción 'L –' si se desea que el compilador trabaje con suficiente rapidez.

Se utiliza esta opción cuando se ha almacenado en cinta una biblioteca de procedimientos y funciones de uso frecuente, que se incluyen en los programas en que se usan.

También es interesante cuando un programa es muy largo e impide la coexistencia del programa fuente y el código objeto en la RAM. Conviene entonces pasar el programa a cinta y llamarlo para compilación. En ese caso, sólo 128 bytes del programa fuente se cargan en la RAM cada vez, lo que deja mucho más espacio para el código objeto.

Esta opción no puede ser anidada.

SECCIÓN 4. EL EDITOR

4.1 Introducción al editor

El editor que proporciona Hisoft Pascal es fundamentalmente un editor de línea, pero aprovecha al mismo tiempo las facilidades de edición de pantalla completa que poseen los ordenadores de tipo MSX.

Para reducir el tamaño del fichero de texto que se crea, el editor realiza cierta compresión de las líneas. El número de blancos al comienzo de la línea se guarda en forma de un carácter y las palabras reservadas se guardan mediante un código de un solo caracter. Esto suele producir una reducción de espacio de un 25%.

El resultado es crear un fichero de texto cuyo almacenamiento es lo más económico posible. A pesar de ello el listado será muy claro si se han utilizado convenientemente los tabuladores.

Cuando comienza la ejecución de HP, se entra directamente en modo de edición. En la pantalla aparece el mensaje

seguido del cursor de edición. Se puede introducir entonces cualquiera de los comandos del editor; estos comandos tienen el formato general siguiente:

C n1,n2,l1,l2 seguido de ⟨RETURN⟩

donde

C es el nombre de uno de los comandos que describiremos a continuación;
n1 y n2 son enteros entre 1 y 32767;
l1 y l2 son cadenas literales con un máximo de 2Ø caracteres cada una;

los números n1 y n2 se deben escribir en forma decimal. Los argumentos se separan con comas, aunque el símbolo separador se puede cambiar con el comando 'Q', como ya veremos. Los espacios en blanco se ignoran, salvo en lo que concierne a las cadenas literales.

Dependiendo del comando, los argumentos pueden ser obligatorios o no serlo. Algunos comandos sólo tienen efecto si se declaran explícitamente determinados argumentos. Otros no exigen la introducción explícita de los argumentos y, por defecto, toman los que se han utilizado en la última ocasión. De entrada, el editor tiene cargados como argumentos el número 1Ø para n1 y n2 y la cadena vacia para l1 y l2.

A cualquier error en la introducción de un comando, responde el editor con el mensaje 'Pardon?', y el comando queda anulado. Que l2 posea más de 2Ø caracteres es considerado un error; sin embargo, si l1 posee más de 2Ø caracteres el editor se limita a ignorar los que sobran.

El nombre del comando es una letra, y es indiferente que se la escriba mayúscula o minúscula.

Durante la introducción de una línea se pueden utilizar los comandos de edición habituales de MSX, como por ejemplo las teclas de movimiento del cursor, ⟨CTRL/F⟩, ⟨CTRL/E⟩, etc.

A continuación vamos a realizar una descripción de cada uno de los comandos del editor. El símbolo '⟨ ⟩' rodeando un argumento indica que el argumento es obligatorio para que el comando tenga efecto.

4.2 Comandos del editor

4.2.1 *Creación e inserción de texto*

Cuando se escribe una línea, ésta se inserta en el fichero de texto en el lugar que le corresponda, el cual viene determinado por el número de línea.

Para insertar líneas de texto basta con escribir un número de línea, dejar un espacio y escribir el texto que se desee, terminando con ⟨RETURN⟩ para que la línea se archive. En cualquier momento se puede abandonar la escritura y volver al nivel de comandos del editor pulsando ⟨CTRL/STOP⟩.

Si se escribe un número de línea seguido de ⟨RETURN⟩ (es decir, una línea sin texto), entonces la línea con ese número es eliminada del fichero de texto (si existía).

La tecla ⟨BS⟩ destruye el caracter anterior al cursor hasta llegar, si se desea, al comienzo de la línea, pero no más allá.

Cuando se pulsa ⟨RETURN⟩, la línea se introduce en un tampón creado por el editor; si la línea consta de más de 8Ø caracteres, se ignoran los que exceden de ese número.

comando I **sintaxis: I n1,n2**
acción: insertar texto

El comando 'I' proporciona automáticamente números de línea, comenzando por el número n1 e incrementándolo cada vez en la cantidad n2. Cuando aparece en la pantalla el número de línea, se puede introducir el texto que se desee, terminando la línea con ⟨RETURN⟩, momento en el que aparecerá un nuevo número de línea. Para terminar se pulsa ⟨CTRL/STOP⟩.

Si se introduce así el número de una línea que ya existe, el editor da a la antigua un nuevo número de línea. Si lo que se desea introducir es un bloque de líneas, convendrá entonces utilizar el valor n2 = 1 para asegurarse de que la inserción es correcta.

Si el incremento automático del número de línea lleva a un número de línea superior a 32767, entonces se termina automáticamente el modo 'inserción' y se pasa al nivel de comandos del editor.

4.2.2 *Listado de texto*

Para ver el contenido del fichero de texto se pueden utilizar los comandos siguientes, que proporcionan un listado del mismo en la pantalla o en la impresora. En el listado, las líneas aparecen tabuladas, separándose claramente los distintos campos de la línea.

comando L **sintaxis: L n1,n2**
acción: listado en pantalla

Este comando tiene por efecto listar en la pantalla las líneas comprendidas entre la n1 y la n2 inclusive. Los valores por defecto de los argumentos son 1 y 32767, respectivamente (no son, pues, los anteriormente usados). Si se utiliza 'L' sin argumentos, se obtiene el listado completo.

El número de líneas que aparecen simultáneamente en la pantalla es de 22; cuando ya hay 22 líneas en pantalla, el listado se detiene. Si aún no se ha llegado a la línea n2, se obtienen otras 22 líneas pulsando cualquier tecla normal. Si se pulsa ⟨CTRL/STOP⟩ se vuelve al nivel de comandos del editor.

comando Z **sintaxis: Z n1,n2**
acción: listado por impresora

Es similar al anterior, pero lanza el listado por la impresora. Si no hay impresora conectada, o si la impresora está desactivada (''off-line''), aparece en pantalla el aviso 'No Printer!' y el comando no tiene efecto. Las teclas ⟨CTRL/STOP⟩ interrumpen el listado y hacen que se vuelva al nivel de comandos.

4.2.3 *Edición de texto*

La mayor parte de las veces es inevitable tener que modificar algunas líneas de texto. Existen varios comandos que permiten corregir, borrar, alterar y renumerar las líneas.

comando E **sintaxis: E n**
acción: edita la línea número n

Edita la línea número n para permitir su modificación. Si la línea n no existe, el comando queda sin efecto. Si existe, la línea se copia en un tampón de la memoria y además aparece en pantalla (número de línea incluido). La línea se puede corregir entonces con ayuda de las flechas de movimiento del cursor y de los comandos de edición del MSX, que podrá usted consultar en el manual de BASIC o en la sección Ø.2. Cuando se hayan concluido las modificaciones, se debe pulsar ⟨RETURN⟩ para validar la línea e insertarla en el fichero de texto.

comando D **sintaxis: D** ⟨n1,n2⟩
acción: borra líneas

Sirve para suprimir del fichero de texto todas las líneas comprendidas entre la n1 y la n2 inclusive. Los argumentos son obligatorios; la ausencia de alguno de ellos o el hecho de que n2 sea menor que n1 dejan sin efecto el comando. Esto se hace para prevenir las catástrofes que pudiera originar algún descuido. Para borrar solamente la línea n, basta hacer n1 y n2 iguales a n, aunque se logra lo mismo escribiendo el número n seguido de ⟨RETURN⟩.

comando M **sintaxis: M** ⟨n1,n2,n3⟩
acción: mueve un bloque de líneas

Hace que el bloque de las líneas que van de la n1 a la n2 inclusive se coloque justo antes de la línea n3, para lo que se renumeran las líneas n3 y posteriores si es necesario. El bloque original se borra. Los tres argumentos son obligatorios.

No existe límite para el tamaño del bloque que se mueve, puesto que las líneas se mueven de una en una.

comando N **sintaxis: N** ⟨n1,n2⟩
acción: renumera las líneas

Renumera el fichero de texto comenzando por el número n1 y dando cada vez un incremento de n2 en el número de línea. Los dos argumentos son obligatorios. Si la renumeración hace que algún número de línea exceda de 32767, entonces se conserva la numeración original.

comando F **sintaxis: F n1,n2,l1,l2**
comando S **sintaxis: S**
acción: búsqueda y sustitución

Los comandos 'F' y 'S' de búsqueda (''find'') y sustitución se usan normalmente combinados. El comando 'F' busca la cadena literal l1 a lo largo de las líneas comprendidas entre la n1 y la n2, recorriendo estas líneas en el sentido de avance del texto. Cuando la encuentra, procede a editar la línea en que aparece la cadena; se puede actuar entonces como en el caso del comando 'E'. Pero también se puede

pulsar la letra 'S', y en tal caso la cadena l1 queda sustituida por la cadena l2. Además, la sustitución produce una nueva búsqueda de la cadena l1. Pulsando repetidamente 'S' y ⟨RETURN⟩ se realiza entonces la búsqueda y sustitución repetida con bastante rapidez.

Cuando no se especifican argumentos en el comando 'F', se toman los empleados en la última ocasión. Esto permite que, para buscar repetidamente la misma cadena, baste con introducir el comando sin parámetros a partir de la segunda vez.

4.2.4 *Grabación y carga de ficheros*

Los comandos 'P' y 'G' sirven para grabar en cinta o leer de la cinta un fichero de texto. El comando 'V' permite comprobar la concordancia entre un fichero de texto y un fichero almacenado en cinta.

comando P **sintaxis: P n1,n2,l1**
acción: graba en cinta un fichero de texto

Graba en cinta el fichero formado por las líneas comprendidas entre la n1 y la n2 inclusive, con el nombre dado por la cadena literal l1. Recuerde que se usarán los argumentos anteriormente empleados si no se proporcionan explícitamente otros. El mensaje

'Press REC/PLAY then any key'

le recordará que debe poner el magnetófono en posición de grabar y pulsar después una tecla cualquiera.

La duración del tono de cabecera del fichero grabado, que por defecto es de 1 segundo aproximadamente, se puede alargar hasta unos 5 segundos si se pone el símbolo '!' como primer carácter del nombre del fichero. Esto suele ser aconsejable cuando posteriormente se va a utilizar este fichero para incluirlo en el texto en memoria.

comando G **sintaxis: G ,,l1**
acción: carga de cinta un fichero de texto

Carga el fichero que haya en la cinta con el nombre dado por la cadena l1 y lo incluye al final del texto que pueda haber en la memoria. Cuando en la memoria exista efectivamente texto, el nuevo texto completo será renumerado con números de líneas que comienzan por el 1 y aumentan de 1 en 1. El mensaje

'Press PLAY then any key'

le recordará que debe poner el magnetófono en posición de lectura.

Cuando el primer carácter del argumento l1 es '!', se ignora este carácter.

comando V **sintaxis: V ,,l1**
acción: verificación de un fichero de texto

Verifica la concordancia entre el fichero de texto de la memoria y el fichero que haya en la cinta con el nombre dado por la cadena l1. Cuando el primer carácter en l1 es '!', se ignora este carácter.

Aparecerá el mensaje 'Verified' o el mensaje 'Failed!' (''fallado'') según que los dos ficheros posean o no el mismo texto.

4.2.5 *Compilación y ejecución*

comando C **sintaxis: C n**
acción: compila el programa

Compila el texto a partir de la línea n; si no se especifica número de línea, la compilación se realiza desde la primera. Léase también la sección Ø.3.

comando R **sintaxis: R**
acción: ejecuta el programa compilado

Ejecuta el código objeto del programa ya compilado, a condición de que no se haya modificado el programa fuente después de la compilación.

comando T **sintaxis: T n**
acción: graba en cinta un código ejecutable

La primera acción del comando es compilar el texto a partir de la línea n (o de la primera, si no se añade el argumento). Si la compilación se realiza sin errores, el comando pregunta: 'Ok?'.

Si su respuesta es 'Y', entonces el efecto es trasladar el código objeto producido por la compilación sobre la tabla de símbolos y el texto fuente; a continuación, las rutinas de ejecución y el código objeto se cargan en la cinta, y el fichero recibe como nombre el literal que se haya utilizado por el comando 'F' en la última ocasión. Finalizada la grabación, el control pasa a BASIC.

Posteriormente se podrá cargar el programa desde la cinta, utilizando el comando BLOAD de BASIC. Por ejemplo, si se ha grabado con el comando 'T' el programa ejecutable PRUEBA, entonces el comando 'BLOAD ''PRUEBA'',R' cargará y ejecutará el programa. Cuando termine la ejecución (o bien si se detecta un error) aparecerá el mensaje 'Run?'. Si su respuesta es 'Y' o 'y' el programa se ejecutará nuevamente; si no, el control volverá a BASIC.

Si su respuesta a 'Ok?' es distinta de 'Y' el control pasa al editor. Éste funcionará sin problemas, ya que en este caso el código objeto no ha sido movido.

La dirección en que el código (movido por el comando 'T') se carga y la dirección más alta que utiliza para almacenar las variables se pueden cambiar mediante el comando 'A'.

comando A **sintaxis: A**
acción: modifica parámetros de la compilación

Permite cambiar tres valores para gestión de la memoria que utiliza el compilador. El comando le comunica los valores actuales (en hexadecimal) y le pregunta si desea otros diferentes. Pulsando ⟨RETURN⟩ se deja como estaba el correspondiente parámetro. Si se introduce un número, el parámetro toma este nuevo valor.

La primera pregunta es

Symbol Table size (Ø5ØØ)?

(el número es un ejemplo) y se refiere al número de bytes en hexadecimal reservados para la tabla de símbolos. Si al compilar usted ha recibido el aviso 'No Symbol Space', deberá reservar una cantidad mayor.

Después se produce la pregunta

Translate Stack (D8ØØ)?

Se refiere a la dirección más alta utilizada por el código (movido con 'T') para el almacenamiento de las variables. Es posible que tenga que disminuir esta dirección si desea cargar además alguna rutina en código de máquina para que sea utilizada por su programa.

La última pregunta será

Translate Start (ØØ8Ø)?

Se refiere a la dirección en que se sitúa el código producido por el comando 'T'. Debe ser siempre más baja que el final del fichero de texto, final que puede ser determinado mediante el comando 'X'.

4.2.6 *Otros comandos*

comando H **sintaxis: H**
acción: pantalla de ayuda

Muestra la pantalla de ayuda, que recuerda cuáles son los comandos del editor.

comando B **sintaxis: B**
acción: pasa a BASIC

Devuelve el control al sistema operativo.

comando Q **sintaxis: Q ,,l1**
acción: cambio de separador de los argumentos

De entrada, el símbolo que delimita los diferentes argumentos de un comando es la coma. El comando 'Q' cambia este delimitador por el primer carácter de la cadena literal l1; este carácter no puede ser un espacio. Una vez que el delimitador se ha cambiado, se debe utilizar siempre el nuevo, incluso en el propio comando 'Q'. El comando 'Y' le dirá cuál es el actual delimitador si se le ha olvidado.

comando Y **sintaxis: Y**
acción: informa sobre los argumentos

Muestra cuáles son los valores actuales del delimitador y de los argumentos n1, n2, l1 y l2. Si va a utilizar en un comando los argumentos por defecto, puede ser interesante comprobar previamente si son los que desea.

comando X **sintaxis: X**
acción: comienzo y final del fichero de texto

Muestra cuáles son las posiciones (en notación hexadecimal) del comienzo y del final del fichero de texto. Mediante este comando se puede saber cuánta memoria queda después del fichero de texto.

comando U **sintaxis: U**
acción: último número de línea

Muestra el número de línea de la última línea del fichero de texto. Es útil para añadir texto al final de un fichero largo.

4.3 Un ejemplo de utilización del editor

Supongamos que, con el comando 'I 10,10', escribe usted el siguiente programa:

```
 10 PROGRAM BUBBLESORT
 20 CONST
 30   Tamano =  100;
 40 VAR
 50   Numeros : ARRAY [1..Tamano] OF REAL;
 60   I       : INTEGER;
 70   Temp : REAL;
 80 BEGIN
 90   FOR I:=1 TO Tamano DO Numero[I] := RANDOM;
100   REPEAT
110     FOR I:=1 TO Tamano DO
120     Nocambios := TRUE;
```

```
130      IF Numero[I] > Numero[I+1] THEN
140        BEGIN
150          Temp := Numero[I];
160          Numero[I] := Numero[I+1];
170          Numero[I+1] := Temp;
180          Nocambios := FALSE
190        END
200    UNTIL Nocambios;
210    FOR I:=1 TO Tamano DO WRITE(Numero[I]:4);
220 END.
```

Leyéndolo más despacio notará que se han introducido algunos errores:

lin 1Ø: falta un ';'
lin 11Ø: se debe poner Tamano-1 en lugar de Tamano;
lin 12Ø: su lugar es la línea 1Ø5;
lin 2ØØ: es Nocambio en lugar de Nocambios.

Además, se ha declarado la variable Numeros, pero luego se utiliza equivocadamente Numero. Finalmente, la variable lógica Nocambio no ha sido declarada. Vamos a describir la secuencia de comandos y operaciones que permite hacer las oportunas rectificaciones.

—Escribir 'F 6Ø,999,Numero,Numeros ⟨RETURN⟩' y utilizar repetidamente el comando 'S'.
—Escribir 'E 1Ø ⟨RETURN⟩ ⟨CTRL/N⟩ ; ⟨RETURN⟩'.
—Escribir 'F 11Ø,11Ø,Tamano, Tamano – 1 ⟨RETURN⟩' y a continuación el comando 'S'.
—Escribir 'E 12Ø ⟨RETURN⟩', mover el cursor a la derecha 3 veces y luego escribir 'Ø5 ⟨RETURN⟩'.
—Escribir 'E 2ØØ ⟨RETURN⟩ ⟨CTRL/N⟩ ⟨BS⟩ ⟨RETURN⟩'.
—Escribir '65 Nocamb : BOOLEAN ;' e introducir la línea con ⟨RETURN⟩.
—Escribir 'N 1Ø,1Ø ⟨RETURN⟩' para renumerar el fichero.

Es muy conveniente que practique usted con el ejemplo que acabamos de exponer, utilizando realmente el editor.

APÉNDICE 1. NÚMEROS Y MENSAJES DE ERROR

A1.1 Números de error generados por el compilador

ERROR 1 Número demasiado grande
ERROR 2 Falta un ';' o un 'END'
ERROR 3 Identificador no declarado
ERROR 4 Falta un identificador
ERROR 5 Utilice '=' en lugar de ':=' en una declaración de constante
ERROR 6 Falta un '='
ERROR 7 Este identificador no puede estar al comienzo de una instrucción
ERROR 8 Falta un ':='
ERROR 9 Falta un ')'
*ERROR*1Ø Tipo incorrecto
*ERROR*11 Falta un '.'
*ERROR*12 Falta un factor
*ERROR*13 Falta una constante
*ERROR*14 Este identificador no es una constante
*ERROR*15 Falta un 'THEN'
*ERROR*16 Falta un 'DO'
*ERROR*17 Falta un 'TO' o un 'DOWNTO'
*ERROR*18 Falta un '('
*ERROR*19 No es válido este tipo de expresión
*ERROR*2Ø Falta un 'OF'
*ERROR*21 Falta una ','
*ERROR*22 Falta un ':'
*ERROR*23 Falta un 'PROGRAM'
*ERROR*24 Falta un parámetro variable
*ERROR*25 Falta un 'BEGIN'
*ERROR*26 Falta una variable en la llamada a READ
*ERROR*27 No se pueden comparar expresiones de este tipo
*ERROR*28 El tipo debe ser INTEGER o REAL
*ERROR*29 No se puede leer este tipo de variable
*ERROR*3Ø Este identificador no es un tipo
*ERROR*31 Falta el exponente en el número real
*ERROR*32 Falta una expresión escalar (no numérica)
*ERROR*33 No son válidas las cadenas literales vací; use CHR(Ø)
*ERROR*34 Falta un '['
*ERROR*35 Falta un ']'
*ERROR*36 El tipo de un índice de vector debe ser escalar
*ERROR*37 Falta un '..'
*ERROR*38 En la declaración de ARRAY falta ']' o ','
*ERROR*39 El límite inferior es mayor que el superior
*ERROR*4Ø Conjunto demasiado grande (con más de 256 elementos)

*ERROR*41 El tipo del resultado de la función debe ser especificado por un identificador de tipo
*ERROR*42 En la declaración de SET falta ']' o ','
*ERROR*43 En la declaración de SET falta '..' o ']' o ','
*ERROR*44 El tipo de parámetro debe ser especificado por un identificador de tipo
*ERROR*45 Un conjunto no puede ser el primer factor en una instrucción que no sea de asignación
*ERROR*46 Falta un escalar (incluido real)
*ERROR*47 Falta un escalar (no real)
*ERROR*48 Conjuntos incompatibles
*ERROR*49 No se pueden comparar conjuntos con '<' o '>'
*ERROR*5Ø Falta alguna de las palabras 'FORWARD', 'LABEL', 'CONST', 'VAR', 'TYPE' o 'BEGIN'
*ERROR*51 Falta una cifra hexadecimal
*ERROR*52 El segundo parámetro en POKE no puede ser un conjunto
*ERROR*53 Vector demasiado grande (>64K)
*ERROR*54 En la definición de RECORD falta un ';' o un 'END'
*ERROR*55 Falta un identificador de campo
*ERROR*56 Falta una variable después de 'WITH'
*ERROR*57 La variable en la instrucción WITH debe ser de tipo RECORD (registro)
*ERROR*58 Identificador de campo no asociado con una instrucción WITH
*ERROR*59 Falta un entero sin signo después de LABEL
*ERROR*6Ø Falta un entero sin signo después de GOTO
*ERROR*61 Esta etiqueta está en un nivel equivocado
*ERROR*62 Etiqueta no declarada
*ERROR*63 El parámetro de la función SIZE debe ser una variable
*ERROR*64 Los punteros sólo se pueden comparar con '=' y '<>'
*ERROR*67 El único formato de escritura para enteros que lleva dos veces el símbolo ':' es 'e:m:H'
*ERROR*68 Una cadena literal no debe contener el carácter de fin de línea
*ERROR*69 El parámetro de NEW, MARK o RELEASE debe ser una variable puntero
*ERROR*7Ø El parámetro de la función ADDR debe ser una variable

A1.2 Mensajes de error durante la ejecución

Cuando se detecta un error durante la ejecución de un programa aparece alguno de los mensajes siguientes, seguido de 'at PC=XXXX', donde XXXX es la posición de memoria en la que ha ocurrido el error. Es posible que la causa del error sea evidente; si no es así, consulte en el listado de compilación las direcciones del código objeto para averiguar en qué parte del programa ocurre el error.

'Halt'	Detención del programa; puede provenir del empleo del procedimiento HALT o de haber pulsado ⟨CTRL/STOP⟩.
'overflow'	Sobrepasamiento de un número.
'Out of RAM'	Desbordamiento de la memoria.
'/ by zero'	División por cero (con / o con DIV).
'Index too low'	Valor del índice demasiado bajo.
'Index too high'	Valor del índice demasiado alto.
'Maths Call Error'	Error en alguna de las funciones matemáticas.
'Number too large'	Número demasiado grande.
'Number expected'	Falta un número.
'Line too long'	Línea demasiado larga.
'Exponent expected'	Falta el exponente.

Cualquiera de estos errores detiene la ejecución.

APÉNDICE 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

HP utiliza los siguientes símbolos con un significado predeterminado:

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

A2.3 Identificadores predefinidos

Los siguientes identificadores se pueden considerar como declarados a lo largo de todo el programa y utilizables en el sentido explicado en la sección 2. Su significado puede cambiar si el usuario vuelve a declararlos en su programa.

```
CONST       MAXINT=32767;

TYPE        BOOLEAN=(FALSE,TRUE);
            CHAR {los 256 caracteres ASCII};
            INTEGER=-MAXINT..MAXINT;
            REAL {los numeros reales representables; vease 1.3}

PROCEDURE   WRITE;WRITELN;READ;READLN;PAGE;HALT;USER;POKE;INLINE;
            OUT;NEW;MARK;RELEASE;TIN;TOUT;RANSEED;

FUNCTION    ABS;SQR;ODD;RANDOM;ORD;SUCC;PRED;INCH;EOLN;
            PEEK;CHR;SQRT;ENTIER;ROUND;TRUNC;FRAC;SIN;COS;
            TAN;ARCTAN;EXP;LN;ADDR;SIZE;INP;
```

APÉNDICE 3. REPRESENTACIÓN Y ALMACENAMIENTO DE DATOS

A3.1 Representación de datos

Detallamos a continuación la manera en que HP representa internamente los datos. La información sobre el tamaño que ocupa cada tipo de dato será de interés para la mayor parte de los programadores; el resto de los detalles es sobre todo importante para mezclar Pascal con código máquina.

A3.1.1 INTEGER

Cada entero ocupa 2 bytes y se representa en la forma de complemento a 2. Cuando el compilador utiliza un registro del Z8Ø para cargar un entero, lo hace en el par HL. Como ejemplos de representación, se tiene

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

A3.1.2 CHAR, BOOLEAN y otros tipos escalares

Todos ellos se almacenan en 1 byte, como enteros binarios sin signo. El compilador utiliza para ellos el acumulador, A, del Z8Ø.

Los caracteres se representan en los 8 bits de acuerdo con el código ASCII; por ejemplo,

```
'E'    =    #45
'['    =    #5B
```

En cuanto a los valores lógicos, TRUE se representa como 1 y FALSE como Ø. Esto hace que ORD(TRUE) = 1 y ORD(FALSE) = Ø.

A3.1.3 REAL

Los números reales se representan internamente a base de mantisa y exponente, como en la notación científica, pero en sistema binario. Como ejemplo, damos la notación científica binaria de algunos números:

```
2 = 2*10⁰    o bien    1.0₂ * 2¹
1 = 1*10⁰    o bien    1.0₂ * 2⁰
```

$-12.5 = -1.25*10^1$ o bien $-25*2^{-1}$

-11001_2*2^{-1}

-1.1001_2*2^3 en forma normalizada

$0.1 = 1.0*10^{-1}$ o bien $\frac{1}{10} = \frac{1}{1010_2} = \frac{0.1_2}{101_2}$

y, dividiendo,

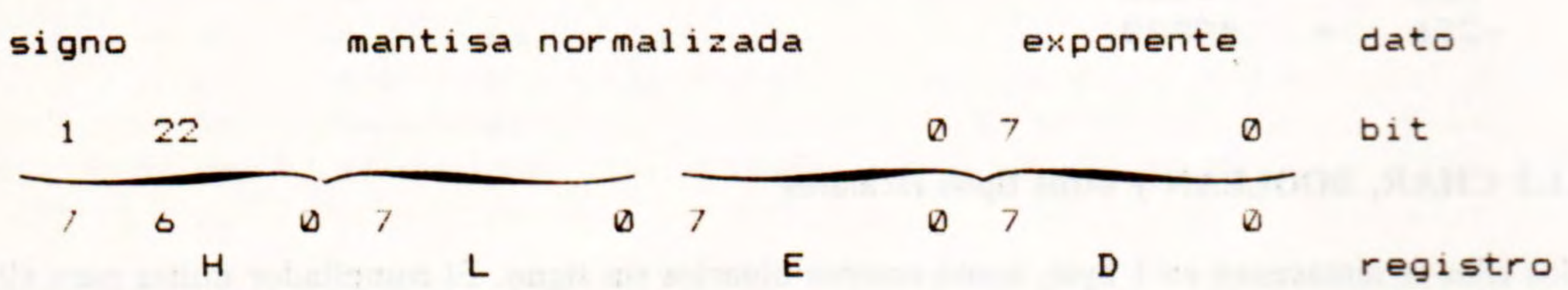
```
                0.0001100
          101  0.1000000000000000
               101
                110
                101
                 1000
                  101
```

(se ve que es periodicamente recursiva)

$$= \frac{0.1_2}{101_2} = 0.0001100_2$$

$1.1001100_2 * 2^{-4}$ en forma normalizada

Se utilizan 4 bytes para almacenar un número real y correspondientemente cuatro registros del Z80: los pares DE y HL. La forma de representación es la siguiente:



Es decir,

signo: el signo de la mantisa (1 = negativo, Ø = positivo) se almacena en 1 bit, el bit 7 de H;

mantisa: se almacena en 23 bits que son, por orden de significación creciente, los Ø...7 de E, los Ø...7 de L y los Ø...6 de H; se utiliza la representación normalizada en la forma 1.xxxxxx; el bit 22 de la mantisa, que es el 6 de H, es siempre 1 (excepto para Ø, que se representa por HL = Ø, DE = Ø);

exponente: se almacena en 8 bits, los de D, en la forma de entero con signo, en representación de complemento a 2.

Los números del ejemplo anterior se almacenarán en los registros en la forma

	reg. H	reg. L	reg. E	reg. D	
2 =	0 1000000	00000000	00000000	00000001	(#40, #00, #00, #01)
1 =	0 1000000	00000000	00000000	00000000	(#40, #00, #00, #00)
-12.5 =	1 1100100	00000000	00000000	00000011	(#E4, #00, #00, #03)
0.1 =	0 1100110	01100110	01100110	11111100	(#66, #66, #66, #FC)

Para realizar la carga de estos números en los registros se necesitarían las siguientes instrucciones del Z8Ø:

```
      2    =   LD   HL,#4000
               LD   DE,#0100

      1    =   LD   HL,#4000
               LD   DE,#0000

   -12.5 =   LD   HL,#E400
               LD   DE,#0300

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

El almacenamiento en memoria de los números reales se realiza en el orden E D L H.

El ejemplo de la representación de Ø.1 permite apreciar cómo un número con una representación decimal exacta puede ser representado con bastante inexactitud en la forma binaria.

A3.1.4 RECORD y ARRAY

Un registro se almacena en la misma cantidad de memoria que el total de sus componentes.

Un vector de n elementos, cada uno de los cuales ocupa s bytes, se almacena en n*s bytes. Por ejemplo, un 'ARRAY[1..1Ø] OF INTEGER' ocupa 2Ø bytes y un 'ARRAY[2..12,1..1Ø] OF CHAR' ocupa 11Ø bytes, puesto que tiene 11*1Ø=11Ø elementos.

A3.1.5 SET

Un conjunto se almacena como una cadena de ceros y unos con tantos elementos como tenga el tipo base. Así, si el tipo base del conjunto es de n elementos, el almacenamiento del conjunto requiere $(n-1)$ DIV $8+1$ bytes. Por ejemplo, un 'SET OF CHAR' requiere $(256-1)$ DIV $8+1=32$ bytes y un 'SET OF (rojo,verde,azul)' requiere $(3-1)$ DIV $8+1=1$ byte.

A3.1.6 Punteros

Los punteros ocupan 2 bytes, que contienen la dirección de la variable a la que apuntan (en formato Intel, o sea, primero el byte bajo).

A3.2 Almacenamiento de las variables durante la ejecución

Se presentan 3 casos diferentes:
- —variables globales, que se declaran en el bloque del programa principal;
- —variables locales, que se declaran en algún bloque interno;
- —parámetros de procedimientos y funciones y valores que entregan las funciones.

Examinaremos los tres casos. Véase también un ejemplo de utilización en el apéndice 4.

Variables globales

Se almacenan desde el comienzo de la pila hacia abajo (hacia direcciones más bajas). Por ejemplo, si la pila está en #BØØØ y las variables del programa principal son

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

las variables se almacenarán en

i	(2 bytes)	#AFFE y #AFFF
ch	(1 byte)	#AFFD
x	(4 bytes)	#AFF9, #AFFA, #AFFB y #AFFC

Variables locales

Al comienzo de cada bloque interno se inicializa el registro IX de tal manera que IX – 4 es la dirección en que empiezan las variables locales del bloque, continuando hacia las posiciones bajas de memoria. Por ejemplo, si el bloque comienza con

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

las variables se almacenarán en

i	(2 bytes)	IX – 6 y IX – 5
j	(2 bytes)	IX – 8 y IX – 7

Parámetros y valores entregados

Cuando los parámetros son valores concretos, se los almacena de manera similar a las variables locales; comienzan en IX + 2 y continúan hacia las posiciones altas de memoria. Por ejemplo, si se tiene

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

los valores x y j se almacenarán en

j	(2 bytes)	IX + 2 y IX + 3 (se almacena antes)
x	(4 bytes)	IX + 4, IX + 5, IX + 6 y IX + 7

Obsérvese que los parámetros que se declaran antes ocupan las posiciones más altas de memoria.

Cuando un parámetro es una variable, el sistema es parecido, pero lo que se almacena es la dirección de la memoria en que se encuentra la variable, ocupando 2 bytes esta dirección. Por ejemplo, si se tiene

```
PROCEDURE test(i:INTEGER; VAR x:REAL);
```

la dirección de x se almacenará en

x	(dirección 2 bytes)	IX + 2 y IX + 3

y el valor i se almacenará en

i	(2 bytes)	IX + 4 y IX + 5

Los valores entregados por las funciones son colocados encima de los parámetros. Por ejemplo, si se tiene

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

el valor i se almacenará en IX + 2 y IX + 3 y se reservará espacio para el valor entregado en IX + 4, IX + 5, IX + 6 y IX + 7.

APÉNDICE 4. EJEMPLOS DE PROGRAMAS

Los programas que siguen pueden ayudarle a despejar posibles dudas que le asalten durante la creación de sus programas en Hisoft Pascal.

```
{Programa para ilustrar el uso de TIN y TOUT. Construye y guarda
en cinta una agenda de telefonos. Usted puede completar el
programa para que efectue busqueda de datos, correcciones, etc.

PROGRAM CINTA;

CONST
  Tamano=10;

TYPE
  Cliente = RECORD
              Nombre : ARRAY [1..10] OF CHAR;
              Numero : ARRAY[1..10] OF CHAR
            END;

VAR
  Agenda : ARRAY [1..Tamano] OF Cliente;
  I : INTEGER;

BEGIN

{Inicializar la agenda}

  FOR I:= 1 TO Tamano DO
  BEGIN
    WITH Agenda[I] DO
    BEGIN
      WRITE('Escriba el nombre');
      READLN;
      READ(Nombre);
      WRITELN;
      WRITE('Escriba el numero');
      READLN;
      READ(Numero);
      WRITELN
  END
END;
```

```
{Para pasar la agenda a cinta utilice ..}

  TOUT('Telefono',ADDR(Agenda),SIZE(Agenda))

{Para cargar de nuevo la agenda utilice ..}

  TIN('Telefono',ADDR(Agenda))

{Ahora puede procesar la agenda como desee}
END.
```

```
 10 {Programa para listar lineas escritas en orden inverso.
 20  Ensena a usar punteros, registros, MARK y RELEASE.}
 30
 40 PROGRAM Vuelvelineas;
 50
 60 TYPE elem=RECORD              {Crea un registro que se apunta a
 70              sig: ^elem;       si mismo}
 80              ch: CHAR;
 90            END;
100       punt=^elem;
110
120 VAR ant,act,mont: punt;    {Punteros de elem}
130
140 BEGIN
150  REPEAT
160    MARK(mont);               {'mont' apunta a la cima del monticulo}
170    ant:=NIL;
180    WHILE NOT EOLN DO
190       BEGIN
200         NEW(act);           {Crea un nuevo registro dinamico
210         READ(act^.ch);       para contener un caracter}
220
230         act^.sig:=ant;      {Cada registro contiene el puntero
240         ant:=act             del registro anterior}
250       END;
260
270 {Escribe la linea al reves, examinando los registros de
280  atras hacia delante.}
290
300    act:=ant;
310    WHILE act <> NIL DO      {NIL es el primero}
320      BEGIN
330         WRITE(act^.ch);     {Escribe el caracter}
340         act:=act^.sig
350      END;
360    WRITELN;
370    RELEASE(mont);           {Libera la memoria de la variable din.}
380    READLN                   {Espera otra linea}
390  UNTIL FALSE                {Use CC para salir}
400 END.
```

```
 10 {Programa para ulilizar la recursion}
 20
 30 PROGRAM FACTOR;
 40
 50 {Calcula el factorial del numero que se escriba en el teclado
 60  1) utilizando la recursion y 2) usando un metodo iterativo}
 70
 80 TYPE
 90   ENPOSI = 0..MAXINT;
100
110 VAR
120   METODO : CHAR;
130   NUMERO : ENPOSI;
140
150 {Algoritmo recursivo}
160
170 FUNCTION RFAC(N : ENPOSI) : INTEGER;
180
190  VAR F : ENPOSI;
200
210  BEGIN
220    IF N>1 THEN F:= N * RFAC(N-1)          {Llama N veces a RFAC}
230           ELSE F:= 1;
240    RFAC := F
250  END;
260
270 {Metodo iterativo}
280
290 FUNCTION IFAC(N : ENPOSI) : INTEGER;
300
310   VAR I,F: ENPOSI;
320   BEGIN
330    F := 1;
340    FOR .I :=2 TO N DO F := F*I;          {Bucle}
350    IFAC:=F
360   END;
370
380 BEGIN
390   REPEAT
400     WRITE('Escriba metodo (I o R) y numero    ');
410     READLN;
420     READ(METODO,NUMERO);
430     IF METODO = 'R'
440          THEN WRITELN(NUMERO,'! = ',RFAC(NUMERO))
450          ELSE WRITELN(NUMERO,'! = ',IFAC(NUMERO))
460   UNTIL NUMERO=0
470 END.
```

```
 10 {Para aprender a hurgar en la memoria, o sea, a modificar las
 20  variables de Pascal utilizando codigo de maquina. Ensena a usar
 30  PEEK, POKE, ADDR e INLINE.}
 40
 50 PROGRAM divmult2;
 60
 70 VAR r:REAL;
 80
 90 FUNCTION divpor2(x:REAL):REAL;       {Funcion que divide por dos
100                                       rapidamente}
110 VAR i:INTEGER;
120 BEGIN
130  i:=ADDR(x)+1;                       {Apunta al exponente de x}
140  POKE(i,PRED(PEEK(i,CHAR)));         {Decrementa el exponente de
150                                       x. Vease apendice 3.1.3}
160  divpor2:=x
170 END;
180
190 FUNCTION mulpor2(x:REAL):REAL;       {Funcion que multiplica por 2
200                                       rapidamente}
210 BEGIN
220  INLINE(#DD,#34,3)                   {INC (IX+3) - el exponente de
230                                       x. Vease apendice 3.2}
240  mulpor2:=x
250 END;
260
270 BEGIN
280  REPEAT
290   WRITE('Escriba el numero r  ');
300   READ(r);                                {No hace falta READLN. Vease
310                                            seccion 2.3.1}
320
330   WRITELN(r dividido por 2 es',divpor2(r):7:2);
340   WRITELN('r multiplicado por 2 es',mulpor2(r):7:2)
350  UNTIL r=0
360 END.
```

BIBLIOGRAFÍA

Las especificaciones sobre Pascal normalizado se pueden encontrar en

K. JENSEN and N. WIRHT
"Pascal User Manual and Report"
Springer Verlag (2nd edition 1978)

Existen en castellano algunos libros sobre Pascal que permiten aprenderlo empezando desde cero. Por ejemplo,

R. ZAKS
"Introducción al Pascal"
Marcombo (1984)

A. K. KELLER
"Programación en Pascal"
McGraw-Hill (1983)

J. P. TREMBLAY, P. R. BUNT y L. M. OPSETH
"Pascal estructurado"
McGraw-Hill (1984)

y también, aunque no trate sólo de Pascal,

N. WITH
"Algoritmos + Estructuras de Datos = Programas"
Ediciones del Castillo (1984)

Producido por:

indescomp

Av. del Mediterráneo, 9. 28007 MADRID.