ABS

Tipo: Función numérica

Sintaxis:

ABS(*expresión numérica*)

Descripción:

Calcula y devuelve el valor absoluto de la expresión numérica indicada.

Ejemplos:

```
PRINT ABS(-12)
PRINT ABS(x*2)
PRINT ABS(x)
```

ACOS

Tipo: Función numérica

Sintaxis:

ACOS(*expresión numérica*)

Descripción:

Calcula y devuelve el arco coseno de la expresión numérica indicada.

Ejemplos:

```
y = ACOS(x)
```

ANSI$

Tipo: Función de cadena

Sintaxis:

ASCII$(*cadena*)

Descripción:

Devuelve la conversión de ASCII/OEM a ANSI de la cadena indicada.

Esta función se utilizará para convertir cadenas que estén codificadas con el juego de caracteres OEM.

La función devolverá la cadena equivalente en Windows ANSI.

Es la función simétrica a ASCII$.

Ejemplos:

PRINT CHR$(&h90) → imprime "É"(con código página 850)
PRINT ANSI$(CHR$(&h90)) → imprime "╔"
PRINT HEX$(ASC(ANSI$(CHR$(&h90)))) → imprime "C9"

ASC

Tipo: Función numérica de cadena

Sintaxis:

ASC(*cadena*)

Descripción:

Devuelve el valor del código ASCII del primer carácter de la cadena indicada.

Si la cadena está vacia, devuelve 0.

Ejemplos:

PRINT ASC("A") → imprime el valor 65

PRINT ASC(A$) → imprime el valor del código ASCII del primer carácter de A$

ASCII$

Tipo: Función de cadena

Sintaxis:

ASCII$(*cadena*)

Descripción:

Devuelve la conversión de ANSI a ASCII/OEM de la cadena indicada.

Esta función se utilizará para convertir cadenas que estén codificadas con el juego de caracteres ANSI.

La función devolverá la cadena equivalente en ASCII OEM, según el código de página del sistema.

Es la función simétrica a ANSI$.

Ejemplos:

PRINT CHR$(&hC1) → imprime "⊥"(con código página 850)
PRINT ASCII$(CHR$(&hC1)) → imprime "Á"
PRINT HEX$(ASC(ASCII$(CHR$(&hC1)))) → imprime "B5"

ASIN

Tipo: Función numérica

Sintaxis:

ASIN(*expresión numérica*)

Descripción:

Calcula y devuelve el arco seno de la expresión numérica indicada.

Ejemplos:

y = ASIN(x)

ATAN

Tipo: Función numérica

Sintaxis:

ATAN(*expresión numérica*)

Descripción:

Calcula y devuelve el arco tangente de la expresión numérica indicada.

Ejemplos:

y = ATAN(x)

AUTO

Tipo: Orden inmediata de edición

Sintaxis:

AUTO
AUTO *número línea*
AUTO *número línea*, *incremento*

Descripción:

Numera automáticamente la creación de nuevas líneas de programa.

AUTO sólo está disponible en programas que dispongan de números de línea.

Al escribir AUTO, sin argumentos, aparece en pantalla el número 10, y el cursor se situa a su derecha, listo para crear la línea 10 de programa. Al introducir dicha línea, se mostrará 20, para crear la siguiente.

AUTO dispone de tres formas de sintaxis:

AUTO: auto-numeración de líneas, comenzando para la 10, usando incrementos de 10.

AUTO *número línea*: la numeración comienza por el número indicado. Se usa un incremento de 10.

AUTO *número línea*, *incremento*: la numeración comienza por el primer número indicado, usándose como incremento el segundo número indicado.

Si la línea auto-numerada ya existe, se mostrará un asterisco "*" para indicarlo. Dicho asterisco no formará parte de la línea.

Para cancelar la auto-numeración, pulsar CTRL-C.

Ejemplos:

AUTO 100 → comenzar numeración desde línea 100
AUTO 100, 5 → comenzar desde línea 100 a incrementos de 5 en 5.

AVG

Tipo: Función numérica

Sintaxis:

AVG(*expresión numérica*, *expresión numérica, expresión numérica …*)

Descripción:

Calcula y devuelve la media aritmética de la lista de expresiones numéricas indicadas (separadas por comas).

En modo matricial, devuelve la matriz de valores medios.

Ejemplos:

PRINT AVG(2, 100, 3*100, -1) → imprime 100.25

BACKCOLOR

Tipo: Variable numérica del sistema

Sintaxis:

BACKCOLOR

Descripción:

Devuelve el valor del color de fondo actual de la pantalla.

Ejemplos:

PRINT BACKCOLOR

BEEP

Tipo: Orden de programa

Sintaxis:

BEEP

Descripción:

Emite un pitido por el altavoz.

BEEP es equivalente a SOUND 1200, 50

Ejemplos:

BEEP

BIN$

Tipo: Función de cadena

Sintaxis:

BIN$(*expresión numérica*)

BIN$(*expresión numérica*, *número dígitos binarios*)

Descripción:

Devuelve la cadena con la representación binaria de la expresión numérica indicada, en forma de texto.

Opcionalmente puede indicarse el número de dígitos binarios (anchura) que se desea obtener como resultado. Si no se especifica, se devolverá la cadena con la anchura mínima.

Los posibles valores para "número de digitos binarios" son: 8, 16, 32

La expresión numérica se interpreta como número entero.

Ejemplos:

PRINT BIN$(10) → imprime 00001010
PRINT BIN$(2010, 32) → imprime 00000000000000000000011111011010

BREAK

Tipo: Orden de programa

Sintaxis:

BREAK

Descripción:

Sale incondicionalmente del bucle actual.

BREAK rompe el bucle actual (de cualquier tipo), continuando la ejecución en la siguiente instrucción fuera de él.

Ejemplos:

```
REM Imprime los números del 1 al 10 (si j >= 10)
FOR i = 1 TO j
   IF i = 10 THEN BREAK
   PRINT i
NEXT
```

BREAKIF

Tipo: Orden de programa

Sintaxis:

BREAKIF

Descripción:

Sale fuera del bloque actual IF-THEN-ELSE-ENDIF.

BREAKIF provoca se ignoren todas las instrucciones siguientes, saltando el bloque IF-ENDIF, reanudándose la ejecución en la siguiente instrucción que esté a continuación de ENDIF.

En caso de usarse dentro de un bucle, éste es cancelado apropiadamente.

BREAKIF sólo es válido para condicionales en bloque.

Ejemplos:

```
REM Imprime los números del -10 al 10
REM Los negativos en rojo
REM Los positivos <= 5 desplazados 4 espacios
REM Los <=10 desplazados 8 espacios
FOR i = -10 TO 10
   IF i <= 5 THEN
```

```
    IF i < 0 THEN COLOR 12:BREAKIF ELSE COLOR 10
    PRINT SPACE$(4);
  ELSE
    PRINT SPACE$(8);
  ENDIF
  PRINT i
NEXT
```

CALL

Tipo: Orden de programa

Sintaxis:

CALL *nombre procedimiento*
CALL *nombre procedimiento* (*expresión, variable, …*
CALL *nombre procedimiento* (*expresión, variable, …*) RETURNS *variable*

CALL *nombre procedimiento@subprograma …*

Descripción:

Realiza una llamada al procedimiento, externo o de usuario, indicado usando los parámetros especificados.

CALL llama al procedimiento y queda a la espera a su finalización.

La llamada puede realizarse a un procedimiento dentro del programa actual, o bien, dentro de un subprograma que esté cargado:

CALL *nombre procedimiento* : llamada dentro del programa actual

CALL *nombre procedimiento@subprograma*: llamada a subprograma

El número de parámetros, y su tipo, han de coincidir con los declarados en PROCEDURE, o en EXTERN PROCEDURE, dependiendo de si el procedimiento es de usuario o externo, respectivamente.

Sólo se permiten expresiones numéricas o de cadena, para los parámetros de entrada al procedimiento.

Los parámetros de salida han de ser obligatoriamente variables, excepto en caso de llamadas a procedimientos externos, en que se puede indicar un valor de cero, cuyo significado es el no utilizar dicho parámetro.

Opcionalmente puede indicarse la variable que almacenará el valor de retorno, indicando para ello la clausula RETURNS. De nuevo, el tipo de la variable debe de coincidir con lo especificado en PROCEDURE.

Si un procedimiento devuelve valor pero éste es llamado sin la clausula RETURNS, dicho valor se perderá.

En caso de llamada a procedimiento de usuario y que sus parámetros de entrada sean matrices, la distinción se realiza en función del modo actual de ejecución (escalar o matricial):

Modo escalar: solo se permiten variables de matrices.
Modo matricial: se permiten expresiones matriciales.

Ejemplos legales:

```
CALL proc1(100*(num+1), salida)
PROCEDURE proc1(IN a, OUT b)

DIM vector(100)
CALL proc2(vector, num, salida)
PROCEDURE proc2(IN a(), INOUT b, OUT c)

SET MODE MATRIX
CALL proc2(MXNRD(3,3) * 10, num, salida)
```

Ejemplos ilegales:

CALL proc1(100, 200) → argumento "b" es de salida. Debe ser una variable

CALL proc2(100, num, salida) → argumento "a()" es matricial. Debe ser una variable matricial

CALL proc2(vector+20, num, salida) → no permitido en modo escalar

CALL proc2(vector, 100, salida) → argumento "b" es de entrada y salida. Debe ser una variable.

En caso de llamadas a subprograma, el procedimiento puede ser de usuario o externo (que allí estén definidos). Esto permite diseñar subprogramas que hagan las funciones de ficheros "interfaz" de librerías de funciones.

Ejemplos:

```
REM Inversión caracteres de una cadena
call invertir("abcdefg", inv$)
print inv$          → imprime "gfedcba"
print call$(invertir3("abcdefg")) → imprime "gfedcba"
end
procedure invertir(in a$, out b$)
  for i = len(a$) to 1 step -1
    inc b$, a$[i]
  next
```

```
endproc
procedure invertir3(in a$) returns b$
   if len(a$) > 1 then b$ = right$(a$, 1) + call$(invertir3(a$[1 .. #len(a$)-1])) \
                    else b$ = a$
endproc

REM Detiene la ejecución del programa 2 segundos
REM Es equivalente a: PAUSE 2000
extern procedure "Sleep", "kernel32.dll", Sleep(in int(4))
call Sleep(2000)

REM Cargar subprograma “interfaz” de Windows
REM Ejecuta la function de Windows “Sleep”
run subprogram “win”, win
call sleep@win(2000)
```

CALL

Tipo: Función numérica

Sintaxis:

CALL(*nombre procedimiento*(*argumentos*))

Descripción:

Llama a un procedimiento devolviendo su valor numérico de retorno.

La función CALL es idéntica a la orden del mismo nombre pero realizando la llamada a nivel de función numérica, lo que permite poder usar procedimientos dentro de expresiones numéricas.

La función CALL devuelve el valor de retorno del procedimiento, o cero si éste no devuelve ningún valor.

Si la función llamada devuelve una cadena, se intenta su conversión a número.

Ejemplos:

```
option base 1
dim a(3)
a(1) = 10 : a(2) = 20 : a(3) = 30
print call(vsumar(a)) → imprime 60
end
procedure vsumar(in vector()) returns suma
   for i = 1 to dbound(vector, 1)
      inc suma, vector(i)
   next
endproc
```

CALL$

Tipo: Función de cadena

Sintaxis:

CALL$(*nombre procedimiento*(*argumentos*))

Descripción:

Llama a un procedimiento devolviendo su valor de cadena de retorno.

CALL$ es la función análoga a CALL para el caso de cadena.

Realiza la llamada al procedimiento indicando, devolviendo su valor de retorno en forma de cadena, o bien cadena vacia, si éste no devuelve ningún valor.

Si la función llamada devuelve un número, éste es convertido a cadena.

Ejemplos:

```
print call$(invertir3("abcdefg")) → imprime "gfedcba"
end
procedure invertir3(in a$) returns b$
   if len(a$) > 1 then b$ = right$(a$, 1) + call$(invertir3(a$[1 .. #len(a$)-1])) \
                  else b$ = a$
endproc
```

CHAIN-NODE-ELSDENODE-ELSECHAIN-ENDCHAIN

Tipo: Orden de programa (estructura de control)

Sintaxis:

```
CHAIN
NODE expresión 1
    .
    .
    ELSE
    .
    .
NODE expresión 2
    .
    .
    ELSE
    .
    .
ELSENODE expresión
    .
```

.
ELSE
.
.
ELSECHAIN
.
.
ENDCHAIN

Descripción:

Ejecuta una serie de instrucciones condicionadas encadenadas.

CHAIN es una estructura equivalente a una secuencia encadenada de estructuras IF-THEN-ENDIF:

```
IF condición 1 THEN
   instrucciones
   IF condición 2 THEN
      instrucciones
      IF condición 3 THEN
      .
      .
```

y también:

```
IF condición 1 THEN
   instrucciones
  ELSE
      IF condición 2 THEN
         instrucciones
      ELSE
            IF condición 3 THEN
            .
            .
```

Los nodos tipo NODE son los equivalentes a IF-THEN y los de tipo ELSENODE a ELSE-IF-THEN.

La estructura CHAIN está compuesta por una serie de nodos (NODE y ELSENODE) que incluyen una condición y un conjunto de instrucciones asociadas.

Todos los nodos de un mismo tipo deben de estar juntos, debiendo de aparecer los ELSENODE después de los NODE.

Las instrucciones de un nodo de tipo “NODE” sólo se ejecutarán si todas las condiciones de los nodos NODE anteriores se cumplieron y además su propia condición se cumple.

Las instrucciones de un nodo de tipo “ELSENODE” sólo se ejecutarán si su condición se cumple y además:

- El anterior nodo es de tipo “NODE”, y fue el primero en no ejecutarse (su condición no se cumplió pero sí todas las de los nodos anteriores)
- El anterior nodo es de tipo “ELSENODE” y su condición no se cumplió.

Un nodo cualquiera puede ir seguido de un bloque ELSE, que se ejecutará en caso de que la condición del nodo no se cumpla y sea el primer nodo en no ejecutarse.

El bloque ELSECHAIN sólo se ejecutará si alguna de las condiciones de los nodos de tipo NODE no se cumplió, o bien, fallaron todos los nodos de tipo ELSENODE.

Ejemplos:

```
‘ Lee una línea seleccionada de un fichero de texto
CHAIN
  NODE TRUE
    FileOpen = FALSE
    INPUT "Número de línea a leer: "; nlinea

  NODE nLinea > 0
  ELSE
    PRINT "Número de línea incorrecto"

  NODE FEXISTS("lista.txt")
    OPEN TEXT READ #1, "lista.txt"
    FileOpen = TRUE
  ELSE
    PRINT "Fichero no existe"

  NODE TRUE
    FOR i = 1 TO nlinea:INPUT #1, linea$:NEXT

  NODE NOT EOF(#1)
    PRINT "Contenido línea: "; linea$
  ELSE
    PRINT "Línea no existe"

ELSECHAIN
  IF FileOpen THEN CLOSE #1
ENDCHAIN
```

CHDIR

Tipo: Orden de programa

Sintaxis:

CHDIR *cadena*

Descripción:

Cambia al directorio indicado en la cadena de caracteres.

Ejemplos:

CHDIR “c:\”

CHECK SYNTAX

Tipo: Orden inmediata

Sintaxis:

CHECK SYNTAX

Descripción:

Realiza una comprobación de sintaxis del programa actual.

En caso de detectarse errores, se mostrará el primero encontrado.

CHECK SYNTAX evita ciertos errores de ejecución del programa, sin embargo no puede detectar aquellos que tengan carácter dinámico.

Ejemplos:

10 PRINT i
20 PRINT j*

CHECK SYNTAX → detectará el error de sintaxis de la línea 20

CHR$

Tipo: Función de cadena

Sintaxis:

CHR$(*expresión numérica*)

Descripción:

Devuelve una cadena con el carácter ASCII correspondiente a la expresión numérica.

CHR$ es la función inversa de ASC

Ejemplos:

PRINT CHR$(65) → imprime la letra “A”

CLEAR

Tipo: Orden de programa

Sintaxis:

CLEAR
CLEAR *variable*
CLEAR *variable*, *variable*, …
CLEAR #*número pila*

Descripción:

Borra el contenido de variables del programa, o bien, vacia una pila de usuario.

CLEAR tiene varias sintaxis para permitir borrar una, varias o todas las variables del programa, y para permitir vaciar una pila de usuario:

CLEAR: borra todas las variables del programa y vacia todas las pilas de usuario.

CLEAR variable: borra la variable indicada

CLEAR variable, variable, … : borra todas las variables indicadas.

CLEAR #*número pila*: vacia la pila de usuario indicada (número entre 0 y 31).

Para el caso de borrado de variables, CLEAR solo actúa sobre el ámbito actual, de forma que, si es ejecutado dentro de un procedimiento, únicamente se borrará la variable dentro del mismo.

Ejemplos:

```
A = 1:B=2
CLEAR A
PRINT A,B → imprime 0 2
CLEAR
PRINT A,B → imprime 0 0

a = 1
CALL borrar
PRINT a  → imprime 1
END
PROCEDURE borrar
```

```
    CLEAR
ENDPROC

PUSH 100
CLEAR #0
POP A → genera un error. La pila #0 está vacia
```

CLIPBOARD$

Tipo: Variable de cadena del sistema

Sintaxis:

```
CLIPBOARD$
CLIPBOARD$ = cadena
```

Descripción:

Devuelve una cadena con el contenido de texto del portapapeles, o bien, lo asigna.

CLIPBOARD$ admite su lectura y escritura.

En caso de lectura y el portapapeles no contenga texto, se devolverá cadena vacia.

Ejemplos:

```
PRINT CLIPBOARD$ → imprime el contenido del portapapeles
CLIPBOARD$ = "texto a copiar"
```

CLOSE

Tipo: Orden de programa

Sintaxis:

```
CLOSE #número fichero
CLOSE #número fichero, #número fichero, …
CLOSE
```

Descripción:

Cierra uno o más fichero, de cualquier tipo.

CLOSE sin ningún argumento, cerrará todos los ficheros actualmente abiertos.

Indicando uno o más números de ficheros, sólo éstos serán cerrados.

Ejemplos:

CLOSE #1
CLOSE #2, #4 → cierra ficheros 2 y 4
CLOSE → cierra todos los ficheros

CLS

Tipo: Orden de programa

Sintaxis:

CLS

Descripción:

Borra la pantalla

CMDLINE$

Tipo: Variable de cadena del sistema

Sintaxis:

CMDLINE$

Descripción:

Devuelve una cadena con el contenido de la línea de comando de ejecución de iBASIC, o del programa compilado.

La primera subcadena de CMDLINE$ es el propio nombre de iBASIC, o del programa compilado.

Ejemplos:

PRINT CMDLINE$ → imprime la línea de comando

CODEPAGE

Tipo: Constante numérica del sistema

Sintaxis:

CODEPAGE

Descripción:

Devuelve el valor del código de página que se usa como mapa de caracteres.

El código de página establece la grafía de los caracteres ASCII.

Ver anexo 4.

COLOR

Tipo: Orden de programa

Sintaxis:

COLOR *primer plano*
COLOR *,fondo*
COLOR *primer plano*, *fondo*

Descripción:

Cambia el color actual (primer plano y/o fondo)

Las distintas sintaxis de COLOR permiten cambiar el color de primer plano (color de letra), el color de fondo, o ambos a la vez.

Los colores están identificados por un número de 0 a 15.

Ver anexo 2 para tabla de colores.

Ejemplos:

COLOR 4 → establece el color de primer plano a rojo
COLOR ,2 → establece el color de fondo a verde
COLOR 12,0 → establece el color de primer plano a rojo brillante y el fondo a negro

COMPILE

Tipo: Orden inmediata

Sintaxis:

COMPILE *fichero ejecutable*
COMPILE *fichero ejecutable*, ICON *fichero icono*
COMPILE *fichero ejecutable*, SUBPROGRAM *fichero subprograma 1, …*
COMPILE *fichero ejecutable*, ICON … , SUBPROGRAM …

Descripción:

Compila el programa actual, generando un fichero ejecutable auto-contenido.

"fichero ejecutable" es una cadena con el nombre del fichero ejecutable de salida que se desea obtener.

Opcionalmente se puede indicar un fichero de icono para ser usado en la aplicación.

Si se utiliza la opción “ICON” y el nombre de fichero del icono es cadena vacia, entonces se el fichero de salida ejecutable se generará sin icono alguno.

Para programas con subprogramas, se pueden especificar los nombres de los ficheros de los subprogramas, de manera que éstos serán incluidos también en el fichero ejecutable final.

De forma previa a COMPILE, se realiza, automáticamente, una comprobación de sintaxis del programa y de los subprogramas (si estos son indicados).

Ejemplos:

COMPILE “nomina” → compila el programa generando un fichero “nomina.exe”

COMPILE “nomina”, SUBPROGRAM “win”,”util” → compila el programa actual incluyendo los subprogramas “win.bas” y “util.bas”

COMPILE “nomina”, ICON “icon02.ico”, SUBPROGRAM “win”, “util”

CONSTANT

Tipo: Orden de programa

Sintaxis:

CONSTANT *variable = valor*
CONSTANT *variable = valor, variable = valor, …*

Descripción:

Crea una o más constantes de programa con el valor indicado.

Una constante es igual a una “variable” excepto en que su valor no puede modificarse.

Usualmente, las constantes se definirán al principio del programa.

Se permiten constantes numéricas y de cadena, pero no arrays.

Ejemplos:

CONSTANT factor = 10, prefijo$ = “pre_”

PRINT 9.2 * factor → imprime 92

COS

Tipo: Función numérica

Sintaxis:

COS(*expresión numérica*)

Descripción:

Calcula y devuelve el coseno de la expresión numérica indicada.

Ejemplos:

```
y = COS(x)
```

CONTINUE

Tipo: Orden de programa

Sintaxis:

CONTINUE

Descripción:

Comienza una nueva vuelta del bucle actual FOR-NEXT, WHILE-WEND, REPEAT-UNTIL, ignorando las órdenes siguientes a CONTINUE.

Ejemplos:

```
FOR i = 1 TO 10
   IF i = 5 THEN CONTINUE
   PRINT i
NEXT
```

Imprime los números 1 al 10, excepto el 5.

CREC

Tipo: Función numérica de fichero

Sintaxis:

CREC(*#numero fichero*)

Descripción:

Devuelve el registro o posición actual del fichero indicado.

Dependiendo del tipo de fichero, la interpretación de CREC es distinta:

Ficheros de texto: devuelve siempre 0

Ficheros binarios raw: devuelve la posición actual en el fichero, comenzando en 1.

Ficheros binarios estructurados: devuelve el número de registro actual, comenzando en 1.

Ficheros multimedia: devuelve el instante de tiempo actual de reproducción, en milisegundos.

Ficheros de red: devuelve el número de bytes leídos.

Ejemplos:

OPEN BINARY #1, “datos.dat”, FIELDS num=INT(4)
GET #1, 90 → leer el registro 90
PRINT CREC(#1) → imprime el valor 91

CURDIR$

Tipo: Variable de cadena del sistema

Sintaxis:

CURDIR$
CURDIR$ = *cadena*

Descripción:

Devuelve una cadena con el directorio actual, o bien, lo establece.

CURDIR$ permite ser usado en modo lectura y escritura.

En caso de asignar un valor, su comportamiento es igual a CHDIR, cambiando al directorio indicado.

Ejemplos:

PRINT CURDIR$ → imprime el directorio actual.

CURDIR$ = “C:\” → establece el directorio actual a “C:\”

CURSOR

Tipo: Orden de programa

Sintaxis:

CUSOR *tamaño*

Descripción:

Establece el tamaño del cursor.

El tamaño es un valor entre 0 a 100 e indica el grado de relleno del cursor.

Un valor 0 oculta el cursor.

Un valor 100 muestra el cursor como un rectángulo relleno.

Ejemplos:

CURSOR 0 → ocultar cursor
CURSOR 50 → Cursor al 50% de tamaño

CVB
CVD
CVI
CVL
CVS

Tipo: Función numérica de cadena

Sintaxis:

CVB(*cadena*)
CVD(*cadena*)
CVI(*cadena*)
CVL(*cadena*)
CVS(*cadena*)

Descripción:

Convierte la cadena, que contiene la representación en memoria de un número, a un número entero o de punto flotante.

Estas funciones son simétricas a MKB$, MKD$, MKI$, MKL$ y MKS$, y se usan en ficheros binarios y de red (conexiones UDP y TCP).

La cadena debe tener una representación de un número, en formato "big endian" o "Little endian", en función del modo actualmente activo en iBASIC.

Hay distintas funciones, según el tamaño y naturaleza del número:

CVB : cadena de 1 byte conteniendo un número entero de 8 bits.

CVI: cadena de 2 bytes conteniendo un número entero de 16 bits.

CVL: cadena de 4 bytes conteniendo un número entero de 32 bits.

CVS: cadena de 4 bytes conteniendo un número de punto flotante de 32 bits.

CVD: cadena de 8 bytes conteniendo un número de punto flotante de 64 bits.

Ejemplos:

PRINT CVB(MKB$(100)) → Imprime 100

SET ENDIAN BIG
READ #1, A$, 4
PRINT CVL(A$) → Imprime el número de 32 bits recibido por el fichero de red (1)

DATA

Tipo: Definición de datos de programa

Sintaxis:

DATA secuencia de números y/o cadenas

Descripción:

DATA se usa para insertar datos literales dentro de un programa, que pueden ser números o cadenas (textos) literales, separados por comas.

La orden DATA, por sí sola, no realiza ninguna función. Necesita de la orden READ, la cual lee un dato almacenado en los DATA.

Los DATA son de carácter local, pudiendo definirlos dentro del ámbito principal del programa, o bien, dentro de los procedimientos.

El uso de las órdenes asociadas READ y RESTORE se limitarán a los DATA presentes en el mismo ámbito de ejecución, esto es, los DATA de un procedimiento no pueden ser accedidos desde otro.

Usualmente, DATA y READ se usan para almacenar datos, muchas veces constantes, del programa.

Ejemplos:

10 READ a$, n
20 PRINT a$, n → imprime cantidad 100
30 DATA “cantidad”, 100

DATALEN

Tipo: Función numérica

Sintaxis:

DATALEN

Descripción:

Devuelve el número total de elementos dentro de los DATA, sean numéricos o de cadena, que estén presentes dentro del ámbito de ejecución actual.

DATALEN es muy útil para limitar el uso de READ y saber de antemano el número máximo de lecturas posibles.

Ejemplos:

10 PRINT DATALEN → imprime 4
20 DATA 10, 2, 4
30 DATA "texto"

DATE$

Tipo: Variable de cadena del sistema

Sintaxis:

DATE$

Descripción:

Devuelve una cadena con la fecha actual.

El formato de la fecha es: DD/MM/AAAA

Ejemplos:

PRINT DATE$ → Imprime 01/12/2009

DBOUND

Tipo: Función numérica

Sintaxis:

DBOUND
DBOUND (*variable*)
DBOUND (*variable, número dimensión*)

Descripción:

Devuelve los límites de un array.

DBOUND se utilizará para obtener los límites, y el número de dimensiones, de una variable de tipo array, sea numérica o de cadena, siendo muy útil para usarse en procedimientos genéricos que traten con matrices de tamaños desconocidos.

Hay tres sintaxis:

DBOUND : devuelve el índice inferior de cualquier array, esto es, el valor establece por OPTION BASE. El número devuelto siempre será 0 ó 1.

DBOUND (*variable*): devuelve el número de dimensiones de la variable indicada. En caso de que la variable no sea un array, se devuelve 0.

DBOUND *(variable, número dimensión*): devuelve el valor del mayor índice de la dimensión y variable indicadas. En caso de que la dimensión no exista, devuelve -1.

Ejemplos:

```
OPTION BASE 0
DIM A(100)
PRINT DBOUND → Imprime 0
PRINT DBOUND (A)  → imprime 1
PRINT DBOUND (A, 1) → imprime 100
PRINT DBOUND (A, 2) → imprime -1

OPTION BASE 1
DIM A(100)
PRINT DBOUND → Imprime 1
PRINT DBOUND (A)  → imprime 1
PRINT DBOUND (A, 1) → imprime 100
```

DEC

Tipo: Orden de programa

Sintaxis:

DEC *variable numérica*
DEC *variable numérica, valor decremento*

Descripción:

Decrementa la variable numérica indicada.

Opcionalmente puede indicarse el decremento a realizar, y en caso de no especificarse, dicho decremento será de 1.

Ejemplos:

```
DEC N  → equivale a: N=N-1
DEC N, 2 → equivale a: N=N-2
DEC A(2), P → equivale a: A(2) = A(2) – P
```

DEF FN

Tipo: Orden de programa

Sintaxis:

DEF FN *nombre función* (*parámetros*) = *expresión función*

Descripción:

Define una función de usuario (numérica o de cadena).

DEF FN define una función, con el nombre indicado, y cuya definición es la expresión indicada a la derecha del signo de igualdad.

La función puede tomar una serie de parámetros de entrada que serán utilizados en la expresión de la función.

Dentro de la expresión se pueden utilizar cualquier tipo de variables, que corresponderán al actual ámbito de ejecución.

Una función puede ser numérica o de cadena en función de su nombre. Si éste finaliza en $ entonces será de cadena, sino será numérica.

La expresión, o cuerpo de la función, será una fórmula cuyo resultado será el valor de retorno de la función.

Una función definida podrá ser usada directamente dentro de cualquier expresión numérica o de cadena indicando su nombre (precedido del prefijo FN) y la lista de argumentos de entrada.

Ejemplos:

```
DEF FNmedia(a,b) = (a+b)/2
PRINT FNmedia(10,2)  → imprime 6
```

DELETE

Tipo: Orden inmediata de edición

Sintaxis:

```
DELETE
DELETE número línea
DELETE desde número línea -
DELETE - hasta número línea
DELETE desde número línea- hasta número línea
```

Descripción:

Borra una o más líneas de programa.

DELETE dispone de varias sintaxis:

DELETE : borra todas las líneas el programa
DELETE *número línea* : borra sólo la línea indicada
DELETE *desde número línea -* : borra todas las líneas desde la indicada.

DELETE - *hasta número línea* : borra todas las líneas hasta la indicada.
DELETE *desde número línea - hasta número línea* : borra el bloque de líneas indicadas

Ejemplos:

DELETE 100 → borra la línea 100
DELETE 500 - → borra todas las líneas desde la 500

DELETE

Tipo: Orden de programa

Sintaxis:

DELETE *variable cadena*, *desde índice*, *número caracteres*

Descripción:

Borra una serie de caracteres de una variable de cadena.

DELETE borra el número de caracteres especificado de la variable de cadena indicada comenzando desde un índice concreto (el carácter del índice también es borrado).

Ejemplos:

A$ = “La casa es grande”
DELETE A$, 9, 3
PRINT A$ → imprime “La casa grande”

DELETE LINE NUMBERS

Tipo: Orden inmediata de edición

Sintaxis:

DELETE LINE NUMBERS

Descripción:

Elimina los números de línea del programa actual, en caso de que los tuviera.

Al realizar la eliminación, se determinan todos los saltos que se realizan con instrucciones GOTO, GOSUB, BREAK y se sustituyen por etiquetas, que son generadas automáticamente.

DEXISTS

Tipo: Función numérica de cadena (de directorio)

Sintaxis:

DEXISTS(*cadena directorio*)

Descripción:

Devuelve TRUE (1) si el directorio que contiene la cadena indicada existe.

Ejemplos:

PRINT DEXISTS("c:\windows") → Imprime 1 si c:\windows existe, sino imprime 0.

DIM

Tipo: Orden de programa

Sintaxis:

DIM *nombre variable (especificación dimensiones), …*

Descripción:

Define uno o más arrays, separados por comas, cuyo nombre es el indicado como "nombre variable" y de las dimensiones especificadas.

La especificación de dimensiones se refiere al índice superior de la dimensión que puede alcanzar el array.

Por defecto, los arrays empiezan con índice 0, excepto que se indique lo contrario con la orden OPTION BASE.

En modo matricial se pueden definir arrays genéricos, sin especificar sus dimensiones. Estos arrays pueden ser usados para guardar resultados de operaciones matriciales, cuyo tamaño se adaptará al del resultado.

Ejemplos:

DIM A(100) → define un vector de 101 elementos (números), desde 0 a 100
DIM B(4, 4) → define una matriz de 5x5 elementos
DIM NOMBRE$(20) → define un vector de 20 cadenas

A(40) = 10 → asigna valor al elemento 40 del vector A
NOMBRE$(0) = "pedro" → asigna valor al elemento 0

SET MODE MATRIX
DIM c
c = MXIDENT(4) → Ahora "c" es una matriz identidad 4x4
SET MODE SCALAR

EDIT

Tipo: Orden inmediata de edición

Sintaxis:

EDIT *número línea*

Descripción:

Edita la línea indicada para su modificación.

En caso de programas sin números de línea, el número es el número de línea físico.

Ejemplos:

EDIT 100 → edita la línea 100

EMPTY

Tipo: Función numérica

Sintaxis:

EMPTY(*cadena*)
EMPTY(#*número pila*)

Descripción:

Devuelve TRUE (1) si la cadena especificada está vacia, o bien, si la pila está vacia.

Para el caso de cadena, EMPTY es equivalente a *cadena = “”*

Ejemplos:

IF EMPTY(A$) THEN PRINT “Cadena vacia”

PUSH 100
PRINT EMPTY(#0) → imprime 0
POP A
PRINT EMPTY(#0) → imprime 1

END

Tipo: Orden de programa

Sintaxis:

END

Descripción:

Finaliza ejecución del programa.

Ejemplos:

IF SALIR =TRUE THEN END

ENDIF

Tipo: Orden de programa

Sintaxis:

ENDIF

Descripción:

Marca el fin de un bloque condicional que haya comenzado por IF-THEN-ELSE.

ENDIF se utilizará solamente en condiciones IF de bloque (multilínea).

Ejemplos:

```
IF A = 10 THEN
   PRINT “A es 10”
   A = 0
ELSE
   PRINT “A no es 10”
ENDIF
```

ENDPROC

Tipo: Orden de programa

Sintaxis:

ENDPROC

Descripción:

Finaliza un procedimiento.

ENDPROC se usa en conjunción con PROCEDURE para la construcción de procedimientos.

Cuando la ejecución encuentra un ENDPROC, finaliza el procedimiento actual, se establecen los valores de las variables de salida, se destruye el ámbito de ejecución del procedimiento y se retorna el control a la orden siguiente al CALL que hizo la llamada.

ENDPROC puede usarse en cualquier punto de un procedimiento, pudiendo aparecer en condiciones u otros puntos.

Ejemplos:

```
REM Invierte cadenzas solo si tienen longitude >= 5 caracteres
call invertir("abc", b1$)
call inverter("abcdefg", b2$)
print b1$  → imprime "abc"
print b2$  → imprime "gfedcba"
end
procedure invertir(in a$, out b$)
   if len(a$) < 5 then b$ = a$ : endproc
   for i = len(a$) to 1 step -1
      inc b$, a$[i]
   next
endproc
```

EOF

Tipo: Función numérica de fichero

Sintaxis:

EOF(*#numero fichero*)

Descripción:

En general, EOF devuelve TRUE (1) si se ha alcanzado el final del fichero especificado:

Ficheros de texto: devuelve TRUE si se alcanzó el fin de fichero

Ficheros binarios: devuelve TRUE si no hay más registros (fin de fichero).

Ficheros multimedia: devuelve TRUE si finalizó la reproducción del fichero multimedia.

Ficheros de red: devuelve TRUE si no hay más datos pendientes de leer.

Ejemplos:

```
INPUT #1, A$
WHILE NOT EOF(#1)
  PRINT A$
  INPUT #1, A$
WEND
```

ERL

Tipo: Variable numérica del sistema

Sintaxis:

ERL

Descripción:

ERL contiene el número de línea en donde ocurrió el último error de programa.

Si no ha habido error, contiene el valor cero.

En caso de programas sin números de línea, ERL contiene el número de línea física.

Habitualmente, ERL se usa en el tratamiento de errores (ON ERROR).

Ejemplos:

PRINT ERL

ERP$

Tipo: Variable de cadena del sistema

Sintaxis:

ERP$

Descripción:

Contiene la cadena del nombre del subprograma que generó el último error de programa.

Si el error fue producido por el programa principal, ERP$ contiene cadena vacia.

ERR

Tipo: Variable numérica del sistema

Sintaxis:

ERR

Descripción:

Contiene el código del último error de programa.

Si no ha habido error, su valor es cero.

Habitualmente, ERR se usa en el tratamiento de errores (ON ERROR).

Ver anexo 1 para tabla de errores.

Ejemplos:

PRINT ERR

ERR$

Tipo: Variable de cadena del sistema

Sintaxis:

ERR$

Descripción:

Contiene la cadena descriptiva del último error de programa (ERR).

Si no ha habido error, su valor es cadena vacia

Ejemplos:

A=)) → genera un error

PRINT ERR$ → imprime “Carácter) no esperado”

EVALUATE

Tipo: Orden de programa

Sintaxis:

EVALUATE *cadena de órdenes*

Descripción:

Evalúa y ejecuta las órdenes BASIC contenidas en la cadena indicada.

Con EVALUATE se pueden ejecutar fragmentos de código dinámicos que pueden ser construidos en tiempo de ejecución.

Dentro de la cadena se permite cualquier conjunto de órdenes BASIC, como si de una nueva línea de programa se tratase.

Ejemplos:

```
INPUT “Orden BASIC a ejecutar:”; a$
EVALUATE a$
```

```
num = 0
EVALUATE “INC num:PRINT num” → imprime 1
```

EVALUATE

Tipo: Función numérica

Sintaxis:

EVALUATE(*cadena órdenes , expresión numérica a devolver*)

Descripción:

Ejecuta la secuencia de órdenes indicada como cadena y devuelve el valor indicado.

Esta es la versión de la orden EVALUATE en forma de función, permitiendo ser utilizada dentro de expresiones.

"E*xpresión numérica a devolver*" es una expresión numérica, cuyo valor se calculará una vez ejecutadas las ordenes de "cadena órdenes", y cuyo resultado será el valor que devuelva EVALUATE.

Ejemplos:

```
‘ Imprime: 1  2  3  4  5  112
PRINT 100 + EVALUATE("for i = 1 to 5 : print i; : next", i * 2)
```

EVALUATE$

Tipo: Función de cadena

Sintaxis:

EVALUATE$(*cadena órdenes , expresión de cadena a devolver*)

Descripción:

Ejecuta la secuencia de órdenes indicada y devuelve la cadena indicada.

EVALUATE$ es la versión de EVALUATE para cadenas, y de igual forma, está pensada para ser usada dentro de expresiones.

Ejemplos:

```
A$ = "La casa es grande"
‘ Imprime: la casa es verde
PRINT EVALUATE$("delete a$, 12, 6", a$) + "verde"
```

EXECUTE

Tipo: Orden de programa

Sintaxis:

EXECUTE *cadena comando*

Descripción:

Ejecuta una acción del sistema operativo.

EXECUTE ejecuta el comando de acción (que está contenido en la cadena) a nivel de interface gráfica.

Para el caso de Windows, se ejecuta de la misma manera a que se haría desde Botón Inicio > Ejecutar …

Normalmente, aparecerá una nueva ventana con resultado del comando.

Por este método se pueden abrir ventanas de exploradores, visualizar imágenes o reproducir archivos multimedia usando la aplicación asociada.

Ejemplos:

EXECUTE "c:\" → abre una ventana de explorador para c:\
EXECUTE "foto01.jpg" → visualiza la imagen utilizando la aplicación asociada.

EXECUTE COMMAND

Tipo: Orden de programa

Sintaxis:

EXECUTE COMMAND *cadena comando*

Descripción:

Ejecuta un comando del sistema operativo.

EXECUTE COMMAND, a diferencia de EXECUTE, ejecuta el comando a nivel de línea de comando (interface modo texto).

Para el caso de Windows, los comandos son los disponibles a nivel de la consola de comandos (cmd), como es el caso de "dir", "del", etc.

Ejemplos:

EXECUTE COMMAND "dir" → obtener el contenido del directorio actual.

EXIT

Tipo: Orden de programa

Sintaxis:

EXIT

Descripción:

Termina el programa y sale de iBASIC, o del programa compilado.

Ejemplos:

IF SALIR = TRUE THEN EXIT

EXP

Tipo: Función numérica

Sintaxis:

EXP(*expresión numérica*)

Descripción:

Calcula y devuelve el valor de $e^{\text{expresión numérica}}$

Ejemplos:

PRINT EXP(1) → devuelve el valor del número "e"

EXTERN PROCEDURE

Tipo: Orden de programa

Sintaxis:

EXTERN PROCEDURE *"nombre","fichero",nombre interno*
(IN *arg, OUT arg, INOUT arg…)*

EXTERN PROCEDURE *"nombre","fichero",nombre interno*
(IN *arg, OUT arg, INOUT arg…) RETURNS arg*

Descripción:

Define un procedimiento externo a iBASIC.

EXTERN PROCEDURE se usará para poder realizar llamadas a procedimientos o funciones localizados en DLLs externas, y particularmente, para invocar a funciones del sistema operativo.

Dentro de la instrucción se define:

- Cadena con el nombre original del procedimiento.
- Cadena con el nombre del fichero de la librería DLL que contiene el procedimiento.
- Nombre interno que se usará en el programa par llamar al procedimiento usando CALL.
- Especificación de los argumentos de entrada.

- Especificación del valor de retorno del procedimiento (opcional).

El nombre original del procedimiento y el interno pueden ser distintos.

Desde el punto de vista de iBASIC, un procedimiento externo como uno definido por el usuario, son similares, y ambos se invocan usando la orden CALL. Es por ello, que no se puede definir un procedimiento externo con el mismo nombre que uno de usuario.

La existencia del fichero de librería DLL y el nombre de procedimiento se realiza en tiempo de ejecución, al usar la orden CALL.

La especificación de la lista de argumentos de entrada es una mezcla a como se hace en PROCEDURE y en la lista de campos de los ficheros binarios (FIELDS).

La especificación de un argumento de entrada tiene el siguiente formato:

dirección tipo_campo(tamaño)
dirección nombre = tipo_campo(tamaño)

siendo:

dirección: indica si el argumento es de entrada o salida:

IN = argumento de entrada
OUT = argumento de salida
INOUT = argumento de entrada y salida

nombre: opcionalmente puede indicarse un nombre descriptivo del argumento. A nivel de ejecución no se utiliza, y su utilidad es de documentación.

tipo campo: indica el tipo de datos y la interpretación que debe darse al campo:

INT = número entero con signo
UINT = número entero sin signo
FLOAT = número de punto flotante
STRING$ = cadena de caracteres o buffer de bytes

tamaño: longitud o tamaño del campo, en bytes. Dependiendo del tipo de campo, se permiten distintos tamaños:

INT = 1, 2 ó 4 bytes (enteros de 8, 16 ó 32 bits)
UINT = 1, 2 ó 4 bytes
FLOAT = 4 ó 8 bytes (números de 32 ó 64 bits)
STRING$ = de 0 a 2^{32} bytes (opcional)

Los números enteros son siempre interpretados en formato “little endian”.

Para argumentos de tipo cadena (STRING$), la especificación de su longitud es opcional y tiene los siguientes efectos:

- Argumentos de entrada: el tamaño es ignorado
- Argumentos de salida: en caso de indicar tamaño, se crea una cadena de dicha longitud para almacenar los datos de salida. Si no se indica tamaño, se usará la longitud de la variable asociada en CALL para realizar la llamada.
- Argumentos de entrada y salida: igual al caso de "solo salida".

En función que tengan argumentos a vectores o a buffers, deberán usarse tipos STRING$ para ello.

Además de la lista de argumentos, de forma opcional puede indicarse el tipo del argumento de salida que tiene la función.

La sintaxis de su especificación es similar al de un argumento normal:

tipo_campo(tamaño)
nombre = tipo_campo(tamaño)

En caso de no indicarse, y si la función devolvería algún valor, éste sería ignorado.

Ver el manual de usuario para más detalles.

Ejemplos:

```
REM Imprime el espacio libre en el disco C: (en bytes)
extern procedure "GetDiskFreeSpaceA"  , "kernel32.dll",
        GetDiskFreeSpace(in string$, out uint(4), out uint(4),
                                out uint(4), out uint(4)) returns uint(4)
call GetDiskFreeSpace("c:\", a, b, c, d)
print a*b*c;"bytes";:print using "(###%)";c/d*100
```

FALSE

Tipo: Constante del sistema

Sintaxis:

FALSE

Descripción:

Devuelve el valor correspondiente a condición falsa, que es cero.

El contrario de FALSE es TRUE

Ejemplos:

IF FALSE THEN PRINT “Esto nunca se imprimirá”

FEXISTS

Tipo: Función numérica de cadena (de fichero)

Sintaxis:

FEXISTS(*cadena fichero*)

Descripción:

Devuelve TRUE (1) si el fichero que contiene la cadena indicada existe.

Ejemplos:

PRINT DEXISTS(“c:\temp\file1.tmp”) → Imprime 1 si el fichero existe, sino imprime 0.

FIELD
FIELD$

Tipo: Campos de ficheros binarios

Sintaxis:

FIELD(#*número fichero, nombre campo numérico*)
FIELD$(#*número fichero, nombre campo cadena*)

Descripción:

FIELD y FIELD$ se utilizan para acceder a los campos de los ficheros binarios (estructurados) abiertos, tanto para obtener su valor, como para asignarlo.

“*nombre campo numérico*” o “*nombre campo cadena*” son los nombres de “variables” indicados en la orden OPEN cuando se abre el fichero binario.

Para **acceder o leer** un campo de un registro de un fichero abierto se procede de la siguiente forma:

GET #*número fichero, número registro*
PRINT FIELD(#*número fichero, nombre campo*)

Para **asignar** un valor a un campo y grabar un registro:

FIELD(#*número fichero, nombre campo*) = *valor a asignar*
PUT #*número fichero, número registro*

Gracias a FIELD y FIELD$, se pueden tener distintos archivos abiertos que usen los mismos nombres de campos.

Ejemplos:

OPEN BINARY #1, “datos.dat”, FIELDS nombre$=STRING$(20), num=INT(4)
GET #1, 1 → leer el primer registro
PRINT FIELD$(#1, nombre$), FIELD(#1, num) → imprime contenido

FIELD$(#1, nombre$) = FIELD$(#1, nombre$) + “*”
FIELD(#1, num) = FIELD(#1, num) + 1
PUT #1, 1 → modifica y graba el registro 1

FOR-TO-STEP

Tipo: Orden de programa

Sintaxis:

FOR *variable numérica* = *expresión desde* TO *expresión hasta*
FOR *variable numérica* = *expresión desde* TO *expresión hasta* STEP *incremento*

Descripción:

FOR, junto con NEXT, forma un conjunto llamado bucle, y es uno de los tipos de bucle soportados por iBASIC.

El conjunto de sentencias que se encuentren entre FOR y NEXT se repetirán un número determinado de veces, en función de lo indicado en la orden FOR.

En la orden FOR se indica una variable numérica, que es el índice del bucle, la cuál, tomará como primer valor el indicado en “*Expresión desde*” y en cada vuelta del bucle se incrementará en un valor dado en “*incremento*” (o en 1, en caso de no indicar la clausula STEP).

El bucle finalizará cuando el índice (variable) de éste sobrepase “*expresión hasta*”.

Dentro de un bucle se pueden crear otros bucles, pero todos ellos deberán tener asociados sus correspondientes NEXT.

Ejemplo:

```
FOR i = 1 TO 10
    PRINT I   → imprime los números desde el 1 al 10
NEXT

FOR i = 10 TO  1 STEP -1
    PRINT I  → imprime los números desde el 10 al 1
NEXT
```

FORECOLOR

Tipo: Variable numérica del sistema

Sintaxis:

FORECOLOR

Descripción:

Devuelve el valor del color de primer plano de la pantalla.

Ejemplos:

PRINT FORECOLOR

FSIZE

Tipo: Función numérica de cadena (de fichero)

Sintaxis:

FSIZE(*cadena fichero*)

Descripción:

Devuelve el tamaño, en bytes, del fichero indicado.

Si el fichero no existe, devuelve 0.

Ejemplos:

PRINT FSIZE("c:\temp\file1.tmp")

GET

Tipo: Orden de programa

Sintaxis:

GET #*numero fichero*
GET #*numero fichero, expresión numero registro*

Descripción:

Lee un registro de un fichero binario estructurado:

GET #*numero fichero*: lee el siguiente registro del fichero
GET #*numero fichero, expresión numero registro*: lee el registro indicado en "*expresión numero registro*".

Al realizarse el GET, el puntero del fichero se sitúa sobre el siguiente registro.

En caso de leerse el último registro del fichero, una llamada a EOF devolvería TRUE.

Ejemplos:

GET #1, 10 → lee el registro 10
GET #1 → lee el registro 11

GOSUB

Tipo: Orden de programa

Sintaxis:

GOSUB *número línea*
GOSUB *número línea@subprograma*

Descripción:

Realiza una llamada a subrutina, saltando a la línea indicada.

GOSUB permite la llamada a subrutinas dentro del programa actual así como a subrutinas de subprogramas cargados previamente:

GOSUB *número línea*: llamada a subrutina en programa actual
GOSUB *número línea@subprograma*: llamada a subrutina en subprograma.

La ejecución volverá al punto actual cuando se encuentre una orden RETURN dentro de la subrutina.

Para el caso de llamada a subprograma, la subrutina es ejecutada dentro del ámbito actual de variables.

Ejemplo:

```
10 GOSUB 100
20 PRINT "fin del programa"
30 END
.
.
100 PRINT "aquí empieza la subrutina"
110 RETURN
```

El programa anterior imprime lo siguiente:

aquí empieza la subrutina
fin del programa

```
REM Llama a la subrutina "calcular" del subprograma "util"
LOAD SUBPROGRAM "rutinas", util
GOSUB calcular@util
```

END

GOTO

Tipo: Orden de programa

Sintaxis:

GOTO *número línea*
GOTO *número línea@subprograma*
GOTO #*número fichero, posición*

Descripción:

Realiza un salto incondicional a una posición del programa/subprograma o salta la reproducción (fichero multimedia) a un punto dado.

La función depende de la sintaxis de GOTO:

GOTO *número línea*: Realiza un salto incondicional a la línea indicada, dentro del programa actual.
GOTO *número línea@subprograma*: Realiza un salto incondicional a la línea indicada, dentro del subprograma indicado (cargado anteriormente).
GOTO *#número fichero, posición*: sólo par ficheros binarios raw o multimedia:

Ficheros binarios raw: posicionarse en la posición indicada, comenzado desde 1 (principio fichero).

Ficheros multimedia: salta a la posición indicada en milisegundos.

En caso de salto a subprograma, una vez finalice éste (con END), el programa actual también finalizará.

Ejemplos:

GOTO 100 → salta a la línea 100
GOTO imprimir_resultados → salta a la etiqueta “imprimir_resultados”

LOAD SUBPROGRAM “prog2”, prog2
GOTO calcular_medias@prog2 → salta a “calcular medias” en subprograma “prog2”

OPEN MEDIA #1, “tema1.mp3”
GOTO #1, 60000 → saltar al minuto 1
PLAY #1 → reproducir.

OPEN BINARY #1, “datos.dat”
GOTO #1, 10
READ #1, a$, 1 → leer décimo byte del fichero
GOTO #1, 1
READ #1, b$ → leer primer byte del fichero

HEX$

Tipo: Función de cadena

Sintaxis:

HEX$(*expresión numérica*)
HEX$(*expresión numérica*, *número dígitos hexadecimales*)

Descripción:

Devuelve la cadena con la representación hexadecimal de la expresión numérica indicada, en formato texto.

Opcionalmente puede indicarse el número de dígitos hexadecimales (anchura) que se desea obtener como resultado. Si no se especifica, se devolverá la cadena con la anchura mínima.

Los posibles valores para “número de dígitos hexadecimales” son: 2, 4, 8

La expresión numérica se interpreta como número entero.

Ejemplos:

PRINT HEX$(10) → imprime 0A
PRINT HEX$(2010, 8) → imprime 000007DA

IF-THEN-ELSE

Tipo: Orden de programa

Sintaxis:

IF *expresión condición* THEN
IF *expresión condición* THEN *sentencias*
IF *expresión condición* THEN *sentencias* ELSE *sentencias*

Descripción:

La construcción IF-THEN-ELSE permite realizar acciones condicionadas en función del cumplimiento de la “*expresión condición*” especificada.

En caso de cumplirse, se ejecutarán las sentencias indicadas después del THEN, y en caso contrario, se ejecutarán las que estén después del ELSE (si está especificado).

IF-THEN-ELSE permite dos tipos fundamentales de construcción:

IF-THEN-ELSE en línea: toda la orden condicional se encuentra en una única línea.

IF-THEN-ELSE de bloque: el condicional puede contener varias líneas físicas del programa, estando comprendido entre el IF inicial y el ENDIF final.

"*Expresión condición*" realmente es una expresión numérica, la cual si tiene como resultado un valor distinto de cero, se considera verdadera, sino será falsa.

La condición puede tener cualquier número de funciones y expresiones numéricas y de cadenas. Así mismo, se soportan expresiones lógicas con los operadores AND, OR, XOR y NOT, de comparación =, >, >=, <, <=, <> y de inclusión "IN []".

Ejemplos:

```
IF a > 0 THEN PRINT "A es mayor de cero" ELSE PRINT "A es cero"
IF a in [1..10, 15] THEN PRINT "A está entre 1 a 10 o vale 15"
IF a = 0 AND b = 0 THEN
   PRINT "A y B valen cero"
ELSE
   PRINT "A y B no valen cero"
ENDIF
```

IF

Tipo: Función numérica

Sintaxis:

IF(*expresión condición , expresión caso verdadero , expresión caso falso*)

Descripción:

Devuelve el valor de la expresión "caso verdadero" o "falso" en función del cumplimiento de la "expresión condición".

Esta función es de carácter condicional, y devuelve un valor u otro según el resultado de la expresión condicional indicada como primer parámetro.

Ejemplos:

A = IF(RND > 0.5, 1, 0) → A = 1 ó 0 de forma aleatoria

IF$

Tipo: Función de cadena

Sintaxis:

IF$(*expresión condición , expresión caso verdadero , expresión caso falso*)

Descripción:

Devuelve la cadena de la expresión “caso verdadero” o “falso” en función del cumplimiento de la “expresión condición”.

IF$ es el caso análogo de la función IF para el caso de cadena.

Ejemplos:

A$ = IF$(RND > 0.5, “on”, “off”) → A$ = “on” u “off” de forma aleatoria

INC

Tipo: Orden de programa

Sintaxis:

INC *variable numérica*
INC *variable numérica, valor incremento*
INC *variable cadena, subcadena*

Descripción:

En general, INC incrementa una variable, pero su funcionamiento particular depende del tipo de variable que le sigue.

Si la variable es numérica, ésta es incrementada con el valor de incremento indicado, o bien, se incrementa en una unidad si el incremento no está indicado.

Si la variable es de cadena, a ésta se le concatena la subcadena (o expresión de cadena) que se indica.

Ejemplos:

INC n → equivale a: n=n+1
INC n, 2 → equivale a: n=n+2
INC a$, b$ → equivale a: a$ = a$ + b$

INKEY$

Tipo: Variable de cadena del sistema

Sintaxis:

INKEY$

Descripción:

Devuelve una cadena con la representación de una tecla pulsada.

INKEY$ devolverá una cadena de 0, 1 ó 2 caracteres con la cadena ASCII de la primera tecla pulsada que aún no se haya procesado.

El uso de INKEY$ implica el tratamiento de dicha tecla, por lo que, un segundo uso de INKEY$ devolverá la segunda tecla pendiente.

En caso de no haber una tecla pulsada, se devuelve cadena vacia (0 caracteres).

Si la tecla es imprimible, se devuelve una cadena con 1 carácter, que correspondiente al código ASCII asociado.

Si la tecla es una especial, de función o de movimiento, se devuelve una cadena de 2 caracteres que representa a la misma.

Ver anexo 3 para tabla de códigos de teclas especiales.

Ejemplos:

```
REM Esperar la pulsación de una tecla
WHILE EMPTY(INKEY$):WEND
```

INPUT

Tipo: Orden de programa

Sintaxis:

INPUT *variable*
INPUT *cadena; variable*
INPUT *cadena, variable*
INPUT INJECT *cadena; variable*
INPUT INJECT *cadena, variable*

INPUT #*numero fichero*, *variable*
INPUT #*numero fichero*, *variable*, *expresión timeout*

Descripción:

En general, INPUT acepta datos del exterior, bien sea del teclado (por el usuario), de un fichero de texto, o de una conexión TCP de red (fichero de red).

El dato que es leído es almacenado en la variable especificada, considerándose el tipo de ésta.

Así pues, si la variable es numérica, el dato a leer debe ser de tal tipo.

Para el caso de entrada de datos desde el teclado, también se permite "inyectar" el contenido actual de la variable a la entrada, usando la clausula INJECT, de forma que aparezca como texto por defecto para el usuario.

Se distinguen tres formatos de INPUT, en función del origen de los datos:

Teclado:

INPUT *variable*: se visualiza el carácter "?" en pantalla y se espera a que el usuario introduzca un dato (de cadena o numérico, en función del tipo de la variable).

INPUT *cadena; variable*: igual al anterior pero visualizando la "cadena" en la pantalla como texto introductorio. También se muestra el "?".

INPUT *cadena, variable*: igual al anterior pero sin visualizar "?".

INPUT INJECT *cadena; variable*
INPUT INJECT *cadena, variable:* igual a los casos anteriores pero "inyectando" el contenido de la variable a la entrada del usuario.

Ficheros de textos:

INPUT #*numero fichero*, *variable*: lee la siguiente línea del fichero de texto, almacenándolo en la variable, considerando el tipo de ésta.

Conexión de red TCP (fichero de red): se utilizará INPUT en conexiones TCP cuyos protocolos de aplicación se basen en texto, como los casos de HTTP o FTP:

INPUT #*numero fichero*, *variable*: esperar y leer la siguiente línea de texto de la conexión y almacenarla en la variable, considerando el tipo de ésta.

INPUT #*numero fichero*, *variable*, *expresión timeout*: igual a la anterior pero esperando un tiempo finito indicado en "timeout". Este tiempo está expresado en milisegundos.

Ejemplos:

INPUT “Introduce un numero: “, num
INPUT “Cual es tu edad”; edad

borrar$=”n”
INPUT INJECT “Borrar el fichero”; borrar$

INPUT #1, línea$, 1000 → leer línea de conexión TCP con 1 segundo de espera.

INSTR

Tipo: Función numérica de cadena

Sintaxis:

INSTR(*cadena, subcadena*)
INSTR(*cadena, subcadena, expresión desde índice*)

Descripción:

Devuelve la posición en donde se encuentra la subcadena dentro de la cadena.

Por defecto, la búsqueda se realiza desde el principio de la cadena, sin embargo, puede indicarse un tercer argumento con el índice desde el cual se debe empezar a buscar.

Si la subcadena no está incluida en la cadena, se devuelve 0.

Ejemplos:

PRINT INSTR(A$ + “texto concatenado”, A$) → Imprime 1
PRINT INSTR(A$ + “texto concatenado”, A$, 2) → imprime 0

INT

Tipo: Función numérica

Sintaxis:

INT(*expresión numérica*)

Descripción:

Devuelve la parte entera de la expresión numérica.

Ejemplos:

PRINT INT(12.56) → imprime 12
PRINT INT(-2.5) → imprime -2

LABEL

Tipo: Orden de programa

Sintaxis:

LABEL *identificador*

Descripción:

Declara una etiqueta, de nombre "*identificador*", en la línea actual de programa.

LABEL solo se permite en programas sin números de línea y se utiliza en órdenes de salto, como GOTO o GOSUB.

LABEL sólo puede estar como primera orden dentro de la línea de programa.

Una forma abreviada de crear etiquetas, es usar el carácter ":" :

etiqueta:

Ejemplos:

```
LABEL bucle_infinito
GOTO bucle_infinito
```

```
REM Imprime los números del 1 al 100
i = 1
contar:
PRINT i
IF i < 100 THEN INC i : GOTO contar
```

LEFT$

Tipo: Función de cadena

Sintaxis:

LEFT$(*cadena, número caracteres*)

Descripción:

Devuelve una subcadena formada por los primeros "*número caracteres*" de la cadena indicada.

Ejemplos:

PRINT LEFT$("primero", 6) → imprime "primer"

LEN

Tipo: Función numérica de cadena

Sintaxis:

LEN(*cadena*)

Descripción:

Devuelve la longitud de la cadena indicada.

Ejemplos:

a$=”Barcelona”
PRINT LEN(a$) → imprime 9

LET

Tipo: Orden de programa

Sintaxis:

LET *variable* = *expresión*

Descripción:

Asigna el resultado de evaluar la “expresión” a la variable.

LET es la orden de asignación, pero su uso es opcional, esto es, se pueden hacer asignaciones sin ella.

Ejemplos:

LET num = 10 → equivale a: num = 10

LET

Tipo: Función numérica

Sintaxis:

LET(*variable numérica = expresión*)

Descripción:

Realiza la asignación del valor a la variable y devuelve el mismo valor.

LET se usará para hacer asignaciones dentro de expresiones numéricas, de manera que asigna y devuelve el valor a dar a la variable.

Ejemplos:

PRINT 100 + LET(A=20+4), A → imprime 124 24

LET$

Tipo: Función de cadena

Sintaxis:

LET$(*variable de cadena = expresión*)

Descripción:

Realiza la asignación a la variable de cadena y devuelve la misma cadena asignada.

LET$ es análogo de LET para el caso de variables de cadenas, y de igual forma, permite hacer asignaciones dentro de expresiones.

Ejemplos:

PRINT "La casa es " + LET$(A$ = "grande") → imprime "La casa es grande"

LIST

Tipo: Orden inmediata de edición

Sintaxis:

LIST
LIST *número línea*
LIST *desde número línea* -
LIST - *hasta número línea*
LIST *desde número línea- hasta número línea*

Descripción:

Lista una o más líneas de programa.

LIST dispone de varias sintaxis:

LIST: lista todas las líneas el programa
LIST *número línea* : lista sólo la línea indicada
LIST *desde número línea -* : lista todas las líneas desde la indicada.
LIST *- hasta número línea* : lista todas las líneas hasta la indicada.
LIST *desde número línea - hasta número línea* : lista el bloque de líneas indicadas

Ejemplos:

LIST -100 → lista las líneas hasta la 100
LIST 100-150 → lista las líneas desde la 100 hasta la 150

LOAD

Tipo: Orden inmediata de edición

Sintaxis:

LOAD *cadena nombre fichero*
LOAD *cadena nombre fichero*, A

Descripción:

Carga el programa almacenado en el fichero indicado, eliminándose el programa actual.

El nombre de fichero puede tener o no extensión. En caso de no especificarla, se asume .BAS.

LOAD detecta automáticamente si el fichero es de texto, binario, o binario protegido.

Para fichero de textos, por defecto se asume que el juego de caracteres es el Windows ANSI (código página 1252).

En caso de que el fichero de texto esté codificado con el juego de caracteres ASCII OEM, deberá utilizarse la opción ",A".

Ejemplos:

LOAD "programa1.bas" → carga programa1.bas
LOAD "c:\programas\programa2.bas" → carga programa2.bas
LOAD "programa3" → carga programa3.bas
LOAD "program4", A → carga el programa usando juego caracteres ASCII OEM.

LOAD SUBPROGRAM

Tipo: Orden de programa

Sintaxis:

LOAD SUBPROGRAM *cadena nombre fichero*, *nombre subprograma*

Descripción:

Carga el subprograma almacenado en el fichero indicado.

El subprograma es cargado e identificado por el "nombre subprograma" especificado, pero no es ejecutado.

Una vez cargado el programa, se podrán llamar a todos los procedimientos de usuario en él definidos, y a todas sus etiquetas o números de línea, usando las órdenes CALL, GOSUB o GOTO.

Sin embargo, en caso de tener declarados procedimientos externos, éstos no podrán ser llamados para que no se ejecuten sus declaraciones.

Estando cargado un programa, éste se puede descargar usando la orden UNLOAD SUBPROGRAM.

Al igual que LOAD, el nombre de fichero puede tener o no extensión, y se detecta automáticamente si el fichero es de texto, binario, o binario protegido.

Ejemplos:

LOAD SUBPROGRAM "utiles.bas", util → Cargar subprograma
CALL convertir_numero@util(100) → llama a un procedimiento de "util".

LOCATE

Tipo: Orden de programa

Sintaxis:

LOCATE fila, columna

Descripción:

Posiciona el cursor (de texto) en la fila y columna indicadas.

Las filas y columnas empiezan en 1.

Para determinar los valores máximos de ambas coordenadas, pueden usarse las variables internas XSIZE e YSIZE.

Ejemplos:

LOCATE 1, 1 → posiciona el cursor en la esquina superior izquierda
LOCATE YSIZE/2, XSIZE/2 → posiciona el cursor en la mitad de la pantalla.

LOG

Tipo: Función numérica

Sintaxis:

LOG(*expresión numérica*)

Descripción:

Calcula y devuelve el logaritmo neperiano (base e) de la expresión numérica indicada.

Ejemplos:

y = LOG(x)

LOWER$

Tipo: Función de cadena

Sintaxis:

LOWER$(*cadena*)

Descripción:

Devuelve la conversión a minúsculas de la cadena indicada.

LOWER$ es la simétrica a UPPER$.

Ejemplos:

PRINT LOWER$("RIO grande") → imprime "rio grande"

MAX

Tipo: Función numérica

Sintaxis:

MAX(*expresión numérica*, *expresión numérica, expresión numérica …*)

Descripción:

Calcula y devuelve el valor máximo de la lista de expresiones numéricas indicadas (separadas por comas).

En modo matricial, devuelve la matriz mayor.

Ejemplos:

PRINT MAX(2, 100, 3*100, -1) → imprime 300

MIN

Tipo: Función numérica

Sintaxis:

MIN(*expresión numérica*, *expresión numérica, expresión numérica …*)

Descripción:

Calcula y devuelve el valor mínimo de la lista de expresiones numéricas indicadas (separadas por comas).

En modo matricial, devuelve la matriz menor.

Ejemplos:

PRINT MIN(2, 100, 3*100, -1) → imprime -1

MKB$
MKD$
MKI$
MKL$
MKS$

Tipo: Función de cadena

Sintaxis:

MKB$(*expresión numérica entero 8 bits*)
MKD$(*expresión numérica punto flotante 64 bits*)
MKI$(*expresión numérica entero 16 bits*)
MKL$(*expresión numérica entero 32 bits*)
MKS$(*expresión numérica punto flotante 32 bits*)

Descripción:

Devuelve una cadena con el contenido de la memoria del número indicando como expresión numérica, es decir, su representación binaria..

Estas funciones tratan los números en formato “Big endian” o “Little endian”, dependiendo del modo actualmente activo en iBASIC, siendo las simétricas de CVB, CVD, CVI, CVL y CVS, en donde cada una de ellas se usará para un tipo de valor diferente, en función de su longitud y de su naturaleza (entero o punto flotante):

MKB : para números enteros de 8 bytes. Devuelve una cadena de 1 carácter.

MKI : números enteros de 16 bits. Devuelve 2 caracteres.

MKL : números enteros de 32 bits. Devuelve 4 caracteres.

MKS : números en punto flotante de 32 bits. Devuelve 4 caracteres.

MKD : números en punto flotante de 64 bits. Devuelve 8 caracters

Su uso está destinado a los ficheros binarios y de red (conexiones UDP y TCP), en donde los valores numéricos vienen empaquetados en un buffer.

Ejemplos:

PRINT CVI(MKI$(100)) → imprime 100

SET ENDIAN BIG
WRITE #1, MKL$(90100) → envía por el fichero de red (1) el número 90100

MKDIR

Tipo: Orden de programa

Sintaxis:

MKDIR *cadena*

Descripción:

Crea el directorio indicado en la cadena de caracteres.

Ejemplos:

MKDIR “c:\temp”

MID$

Tipo: Función de cadena

Sintaxis:

MID$(*cadena, índice inicial, número caracteres*)

Descripción:

Devuelve una subcadena, de longitud máxima “número caracteres”, extraída desde el índice inicial de la cadena que se indica.

Esta función se usará para obtener subcadenas que están en medio de la cadena.

Si la cadena es una variable, una forma equivalente abreviada es utilizar el operador [].

Ejemplos:

PRINT MID$(“primero”, 4, 3) → imprime “mer”

A$=”primero”
PRINT MID$(A$, 4, 3) → equivale a: PRINT A$[4..6]

MXCOF

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXCOF(*expresión matricial*)

Descripción:

Calcula y devuelve la matriz de cofactores de la expresión matricial indicada.

Esta función solo es válida para matrices de dimensión 2.

Ejemplos:

```
DIM c
c = MXCOF(a + b) → calcula la matriz de cofactores de (a+b)
```

MXCOLUMNS

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXCOLUMNS(*expresión matricial*)

Descripción:

Devuelve el número de columnas de la expresión matricial indicada.

Esta función solo es válida para matrices de dimensión 1 ó 2.

Ejemplos:

```
DIM c
c = MXIDENT(5)
PRINT MXCOLUMNS(c) → imprime 5
```

MXDET

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXDET(*expresión matricial*)

Descripción:

Calcula y devuelve el determinante de la expresión matricial indicada.

Esta función solo es válida para matrices cuadradas.

Ejemplos:

```
DIM c
c = MXRND(4, 4)
PRINT MXDET(c) → imprime el determinante de la matriz aleatoria "c"
```

MXDIM

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXDIM(*expresión matricial*)

Descripción:

Devuelve el número de dimensiones de la expresión matricial.

Ejemplos:

```
DIM c
c = MXIDENT(5)
PRINT MXDIM(c) → imprime 2
```

MXIDENT

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXIDENT(*orden matriz*)

Descripción:

Devuelve una matriz identidad (cuadrada) del orden indicado.

Ejemplos:

```
DIM c
c = MXIDENT(3) → c = matriz identidad de orden 3
```

MXINV

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXINV(*expresión matricial*)

Descripción:

Calcula y devuelve la matriz inversa de la expresión matricial indicada.

Esta función sólo es válida para matrices cuadradas.

En caso de que el determinante sea cero (matriz inversa no existe), se genera un error de “división por cero”.

Ejemplos:

```
DIM c, d
c = MXRND(4, 4)
d = MXINV(c)
```

MXISUM

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXISUM(*expresión matricial*)

Descripción:

Calcula y devuelve la suma de todos los elementos de la expresión matricial indicada.

Ejemplos:

```
DIM c
c = MXIDENT(3)
PRINT MXISUM(c) → imprime 3
```

MXIPROD

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXIPROD(*expresión matricial*)

Descripción:

Calcula y devuelve el producto de todos los elementos de la expresión matricial.

Ejemplos:

```
DIM c
c = MXIDENT(3)
PRINT MXIPROD(c) → imprime 1
```

MXONES

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXONES(*especificación de dimensiones*)

Descripción:

Devuelve una matriz con un número de dimensiones y tamaño indicados, cuyos elementos tienen todos ellos valor 1.

Ejemplos:

DIM c, d
c = MXONES(6, 6) → c = matriz cuadrada 6x6 con sus elementos a 1
d = MXONES(6, 6) * 4 → d = matriz cuadrada 6x6 con sus elementos a 4

MXORDER

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXORDER(*expresión matricial*)

Descripción:

Devuelve el orden de la expresión matricial.

Esta función solo es válida para matrices cuadradas.

Ejemplos:

PRINT MXORDER(MXIDENT(2)) → imprime 2

MXPROD

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXPROD(*expresión matricial a, expresión matricial b*)

Descripción:

Calcula y devuelve el producto vectorial de las expresiones “a” por “b”.

Las matrices han de ser compatibles a nivel de producto para poder realizar la operación.

En general, MXPROD (a, b) es distinto a: a*b

Ejemplos:

DIM c
c = MXRND(4, 4)
PRINT MXPROD(c, MXINV(c)) → imprime la matriz identidad 4x4

MXRND

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXRND(*especificación de dimensiones*)

Descripción:

Devuelve una matriz con un número de dimensiones y tamaño indicados, cuyos elementos tienen todos ellos valores aleatorios entre 0 y 1.

Ejemplos:

DIM c, d
c = MXRND(6, 6) → c = matriz cuadrada 6x6 con sus elementos aleatorios entre 0 y 1 (decimales)
d = ROUND(MXRND(6, 6) * 10) → d = matriz cuadrada 6x6 con sus elementos aleatorios entre 0 y 10 (enteros)

MXROWS

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXROWS(*expresión matricial*)

Descripción:

Devuelve el número de filas de la expresión matricial indicada.

Esta función solo es válida para matrices de dimensión 1 ó 2.

Ejemplos:

DIM c
c = MXIDENT(5)
PRINT MXROWS(c) → imprime 5

MXTRACE

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXTRACE(*expresión matricial*)

Descripción:

Calcula y devuelve la traza (suma de los elementos de la diagonal) de la expresión matricial.

Esta función solo está definida para matrices cuadradas.

Ejemplos:

PRINT MXTRACE(MXIDENT(4)*3) → imprime 12

MXTRANS

Tipo: Función matricial (solo en modo matricial)

Sintaxis:

MXTRANS(*expresión matricial*)

Descripción:

Calcula y devuelve la matriz transpuesta de la expresión matricial indicada.

Esta función sólo es válida para matrices de 1 ó 2 dimensiones.

Ejemplos:

```
DIM c, d
c = MXRND(4, 4)
d = MXTRANS(c) → matriz transpuesta de "c"
```

NEG

Tipo: Función numérica

Sintaxis:

NEG(*expresión numérica*)

Descripción:

Calcula y devuelve el complemento a uno de la expresión numérica.

La expresión numérica se interpreta como número entero, invirtiendo sus bits de valor.

Ejemplos:

PRINT HEX$(NEG(&b11110001)) → imprime 00001110

NEW

Tipo: Orden de programa

Sintaxis:

NEW

Descripción:

Borra el programa actual y todas las variables.

Típicamente NEW se usará únicamente en modo interactivo.

NEXT

Tipo: Orden de programa

Sintaxis:

NEXT
NEXT *índice bucle*
NEXT *índice bucle, índice bucle, …*

Descripción:

NEXT establece el final de un bucle FOR-NEXT y hace que éste realice una nueva vuelta si aún, la variable índice del bucle, no ha llegado a su valor final.

Opcionalmente puede indicarse, junto a NEXT, la variable índice del bucle, la cual debe de coincidir con la especificada en la orden FOR.

También es posible indicar en NEXT más de un índice de un bucle, para así cerrar varios que estén anidados, debiéndose especificar tales índices en orden inverso al de aparición de los bucles FOR.

Ejemplos:

```
FOR i = 1 TO 10:PRINT i:NEXT
FOR j = 10 TO 1 STEP -1:PRINT i:NEXT j
FOR i = 1 TO 5:FOR j = 1 TO 2:PRINT i*j:NEXT j, i
```

NOP

Tipo: Orden de programa

Sintaxis:

NOP

Descripción:

NOP es una orden sin contenido y no realiza ninguna acción.

Se puede utilizar en condiciones IF como alternativa para no tener que negar toda la condición.

Ejemplos:

IF a = 0 THEN NOP ELSE PRINT “a no es cero”

NREC

Tipo: Función numérica de fichero

Sintaxis:

NREC(*#numero fichero*)

Descripción:

Devuelve el número total de registros, longitud o duración del fichero indicado.

Dependiendo del tipo de fichero, la interpretación de NREC es distinta:

Ficheros de texto: devuelve siempre 0

Ficheros binarios raw: devuelve el número total de bytes del fichero.

Ficheros binarios estructurados: devuelve el número total de registros. Si el fichero está vacio, devuelve 0.

Ficheros multimedia: devuelve la duración, en milisegundos, del contenido multimedia.

Ficheros de red: devuelve el número de bytes recibidos pendientes de ser leídos.

Ejemplos:

OPEN BINARY #1, “datos.dat”, FIELDS num=INT(4)
PRINT NREC(#1) → imprime el número de registros de datos.dat

ON ERROR

Tipo: Orden de programa

Sintaxis:

ON ERROR CONTINUE
ON ERROR GOSUB *número línea*
ON ERROR GOTO *número línea*
ON ERROR STOP

Descripción:

ON ERROR se utiliza como mecanismo de tratamiento de errores en tiempo de ejecución.

Típicamente los errores serán de tipo sintáctico del lenguaje BASIC, o bien por el uso de argumentos que estén fuera de rango.

En caso de no especificar una orden ON ERROR, en caso de producirse algún error, la ejecución del programa se detiene, visualizándose un texto descriptivo del problema y la línea en dónde ocurrió.

Hay 4 formas de tratamiento de error:

ON ERROR CONTINUE: en caso de error, la ejecución continuará en la siguiente orden.
ON ERROR GOSUB *numero línea*: en caso de error, se realiza una llamada a la subrutina indicada. El retorno de la subrutina devolverá el control a la siguiente orden que causó el problema.
ON ERROR GOTO *numero línea*: en caso de error, se realiza un salto a la línea (o etiqueta) indicada.
ON ERROR STOP: en caso de error, se detiene la ejecución del programa.

ON ERROR se puede utilizar más de una vez dentro del programa, aunque normalmente sólo aparecerá en las primeras líneas de éste.

ON ERROR también se utiliza junto a ERR y ERL para determinar cuál fue el error producido y la línea en dónde sucedió.

En caso de una estructura con subprogramas, ON ERROR sólo se aplica al programa principal, de manera que si surge un error en la ejecución dentro de un subprograma, éste será tratado según la acción de ON ERROR definida en el programa principal.

Ejemplos:

```
ON ERROR GOTO tratamiento_errores
```

ON GOSUB
ON GOTO

Tipo: Orden de programa

Sintaxis:

ON *expresión numérica* GOSUB *número línea, número línea, número línea …*
ON *expresión numérica* GOTO *número línea, número línea, número línea …*

ON *expresión numérica* GOSUB *número línea@subprograma ..*
ON *expresión numérica* GOTO *número línea@subprograma ..*

Descripción:

Estas órdenes realizan un salto a subrutina, o un salto incondicional, respectivamente, en función del valor que resulte de la expresión numérica indicada.

Detrás de GOTO/GOSUB se especificarán tantos números de línea (o etiquetas) de salto como posibles valores pueda tomar la expresión numérica, siendo el primer número de línea el correspondiente al valor 1, el segundo al valor 2 y así sucesivamente.

Los saltos se pueden hacer dentro del programa actual, o bien, a un subprograma dado (indicándose con @):

número línea@subprograma : realiza el salto a la línea (o etiqueta) del subprograma indicado.

En caso de no existir número de línea para el valor resultante, la ejecución continuará con la siguiente orden.

Ejemplos:

```
REM si a = 1, saltar a 100
REM si a = 2, saltar a 200
REM si a = 3, saltar a 300
ON a GOTO 100, 200, 300
```

```
REM si a = 4, salta a línea 100 de subprograma “prog2”
ON a GOTO 100, 200, 300, 100@prog2
```

OPEN BINARY

Tipo: Orden de programa

Sintaxis:

OPEN BINARY #*número fichero*, *nombre fichero*
OPEN BINARY #*número fichero*, *nombre fichero*, FIELDS *especificación campos*
OPEN BINARY READ #*número fichero, …*

Descripción:

Abre un fichero de tipo binario, en modo raw (sin estructurar), o estructurado en registros de longitud fija.

Al fichero abierto se le asocia el número de fichero indicado, que se usará posteriormente con las órdenes de manejo (GET, PUT, READ, WRITE, …).

El fichero permanecerá abierto hasta que no se invoque la orden CLOSE con el número de fichero.

Opcionalmente, un fichero puede abrirse de modo “sólo lectura”, especificando la clausula READ. En este modo, además, el fichero debe existir.

En caso de no indicar READ y de no existir el fichero, éste es creado.

Fichero raw (no estructurados):

Un fichero abierto en modo raw es aquel sobre el que se puede hacer operación de lectura/escritura en cualquier punto del mismo, usando cualquier longitud.

La sintaxis de apertura del fichero es:

OPEN BINARY #*número fichero*, *nombre fichero*

Con las órdenes READ y WRITE se realizan las operaciones de lectura y escritura, respectivamente, y con GOTO la de salto a un punto concreto dentro del mismo.

Ficheros estructurados:

Un fichero estructurado es aquel que tiene un formato o estructura interna, consistente en un conjunto de registros de tamaño fijo.

La sintaxis de apertura del fichero es:

OPEN BINARY #*número fichero*, *nombre fichero*, FIELDS *especificación campos*

La especificación de campos define la estructura de cada registro del fichero, y es una secuencia de uno más campos, separados por comas, en donde se indica el nombre, tipo y longitud de cada uno ellos.

El formato de la especificación de campos es el siguiente:

nombre campo = *tipo campo* (*tamaño*), *nombre campo = …*

siendo:

nombre campo: variable o nombre del campo del registro, que es tratado como si de una variable del programa se tratase, pero a nivel local del fichero. Los campos de cadena, o de buffer, tendrán el carácter final “$” .

tipo campo: indica el tipo de datos y la interpretación que debe darse al campo:

INT = número entero con signo
UINT = número entero sin signo
FLOAT = número de punto flotante
STRING$ = cadena de caracteres o buffer de bytes

tamaño: longitud o tamaño del campo, en bytes. Dependiendo del tipo de campo, se permiten distintos tamaños:

INT = 1, 2 ó 4 bytes (enteros de 8, 16 ó 32 bits)
UINT = 1, 2 ó 4 bytes
FLOAT = 4 ó 8 bytes (números de 32 ó 64 bits)
STRING$ = de 1 a 2^{32} bytes

Los números enteros son almacenados según el modo configurado en iBASIC (little endian o big endian).

El acceso a los campos, tanto para leer como para asignar, se realiza usando las órdenes FIELD y FIELD$, según el campo sea numérico o de texto (string).

Con las órdenes GET y PUT se realiza la lectura y escritura de registros, respectivamente.

Para acceder a ficheros binarios sin organización en registros, se puede abrir

Como alternativa a la especificación de campos de FIELDS, se puede especificar un único campo de tipo STRING$ que contenga todo el registro, a modo de buffer, y posteriormente se extraigan sus campos manualmente, usándose órdenes de cadena y las funciones CV* / MK*$.

Ejemplos:

```
OPEN BINARY #1, "datos.dat", FIELDS nombre$=STRING$(50),
edad=UINT(1),fecha_alta$=STRING$(10)
REM Grabar primer registro
FIELD$(#1, nombre$) = "Federico López"
FIELD(#1, edad) = 40
FIELD$(#1, fecha_alta$) = DATE$
PUT #1, 1
REM Leer el ultimo registro
GET #1, NREC(#1)
PRINT "Nombre: "; FIELD$(#1, nombre$);" edad: "; FIELD(#1, edad)
CLOSE #1
```

```
OPEN BINARY #1, "datos2.dat"
GOTO #1, 10  ' Ir al byte número 10
READ #1, a$, 20  ' Leer 20 bytes desde posición 10
PRINT a$
CLOSE #1
```

OPEN MEDIA

Tipo: Orden de programa

Sintaxis:

OPEN MEDIA #*número fichero*, *cadena nombre fichero*

Descripción:

Abre un fichero de tipo multimedia, para reproducción de sonido o de video.

El fichero se abre en modo lectura y debe de existir, además, debe ser un fichero de audio o de video que esté soportado por el sistema operativo.

Una vez abierto el fichero, se podrá proceder a su reproducción, de forma síncrona o asíncrona, usando la orden PLAY.

Las funciones CREC, NREC y SREC siempre devuelve un número, expresado en milisegundos.

Ejemplos:

OPEN MEDIA #1, "tema1.mp3"
PLAY #1 → reproduce el sonido asíncronamente
WHILE … → esto se ejecuta mientras el sonido se reproduce.
PAUSE #1 → detener reproducción.

OPEN MEDIA #2, "video1.avi"
PLAY #2, WAIT → reproduce el video síncronamente (en ventana)

OPEN NETWORK

Tipo: Orden de programa

Sintaxis:

OPEN NETWORK UDP(*puerto)* #*número fichero*, *cadena host*
OPEN NETWORK TCP(*puerto)* #*número fichero*, *cadena host*

Descripción:

Abre una conexión de red, bajo protocolo UDP ó TCP, dirigido al host y puerto indicados.

Para protocolo UDP, realmente no se hace una conexión, sino una vía para recibir y enviar tramas desde y hacia el host.

El programa actual actúa como cliente.

Para la comunicación, se utilizan las órdenes READ y WRITE para recibir y enviar datos.

Para TCP, y en el caso de que el protocolo de aplicación sea en modo texto, como HTTP, se podrán usar las órdenes INPUT y PRINT.

La función SREC resulta de interés, ya que devuelve el número de bytes recibidos pero aún no leídos con READ.

Usualmente, en conexiones UDP/TCP, se usarán buffers como estructura de datos a enviar, y que serán los mensajes, tramas o paquetes de la comunicación.

En iBASIC, un buffer es una cadena de caracteres, y la extracción y creación de valores numéricos se hará por medio de las funciones CV* y MK*$, que pueden usar organización “Little endian” o “Big endian”, según esté configurado iBASIC.

De manera estándar, los números enteros, en TCP/IP, deben estar en formato “big endian”, por lo que será necesario cambiar el modo de iBASIC usando la orden “SET ENDIAN BIG”.

Ejemplos:

```
SET ENDIAN BIG
OPEN NETWORK TCP(80) #1, “www.sitioweb.com”
PRINT #1, “GET /index.html”:PRINT #1 → envia comando para obtener index.html
INPUT #1, a$
PRINT a$ → imprime la respuesta.
CLOSE #1
```

OPEN TEXT

Tipo: Orden de programa

Sintaxis:

OPEN TEXT APPEND #*número fichero*, *cadena nombre fichero*
OPEN TEXT CREATE #*número fichero*, *cadena nombre fichero*
OPEN TEXT READ #*número fichero*, *cadena nombre fichero*

Descripción:

Abre un fichero de texto.

Los ficheros de texto son aquellos que están constituidos por líneas de texto, terminadas con los caracteres CR y/o LF.

La apertura puede hacerse de tres modos:

APPEND: se abre el fichero, en modo escritura, para añadir textos al final del mismo.

CREATE: se abre, en modo escritura, pero creándose el fichero. En caso de existir, éste es truncado, y toda su información se pierde.

READ: se abre el fichero en modo lectura, posicionándose el puntero de lectura al principio del mismo. El fichero debe existir.

Como se observa, un fichero de texto no puede abrirse para realizar lectura y escritura simultáneamente.

Se usarán las órdenes INPUT e PRINT para leer y escribir datos, las cuáles tratan los caracteres CR/LF de terminación.

Ejemplos:

```
OPEN TEXT CREATE #1, "datos.txt"
PRINT #1, "línea de texto"
CLOSE #1
```

```
OPEN TEXT READ #1, "datos.txt"
INPUT #1, a$
PRINT a$ → imprime la primera línea del fichero
CLOSE #1
```

OPTION BASE

Tipo: Orden de programa

Sintaxis:

```
OPTION BASE 0
OPTION BASE 1
```

Descripción:

Establece el índice inicial de acceso a los arrays.

Sólo se permiten dos valores, 0 y 1, y la inclusión de esta orden debe hacerse una sola vez en el programa, y antes de usar una orden DIM.

Por defecto, se utiliza el valor 0 como índice inicial.

Ejemplos:

```
OPTION BASE 0
DIM a(3) → elementos: a(0), a(1), a(2), a(3)
```

```
OPTION BASE 1
DIM a(3) → elementos: a(1), a(2), a(3)
```

PAUSE

Tipo: Orden de programa

Sintaxis:

PAUSE *milisegundos*
PAUSE #*número fichero*

Descripción:

PAUSE tiene dos interpretaciones distintas:

PAUSE *milisegundos*: realiza una pausa de la ejecución del programa del número de milisegundos indicado.
PAUSE #*número fichero*: pone en pausa la reproducción del fichero multimedia indicado. Posteriormente, para reanudar la reproducción, usar la orden PLAY.

Ejemplos:

PAUSE 1000 → detiene la ejecución durante 1 segundo

```
OPEN MEDIA #1, "tema1.mp3"
PLAY #1
PAUSE #1 → pausa la reproducción
```

PI

Tipo: Constante del sistema

Sintaxis:

PI

Descripción:

Devuelve el valor del número PI (3.14159265)

PLAY

Tipo: Orden de programa

Sintaxis:

PLAY #*número fichero*
PLAY #*número fichero, opciones*

Opciones disponibles:

REPEAT
WAIT
FULL SCREEN
WINDOW *tamaño x, tamaño y*
LOCATE *x, y*
TITLE *cadena título*

Descripción:

Reproduce, o reanuda, el fichero multimedia indicado.

Opcionalmente, pueden añadirse una o más opciones, separadas por comas:

REPEAT: realiza un bucle de la reproducción, de forma que, al finalizar se comience de nuevo.
WAIT: reproducción síncrona. El programa se detiene hasta que la reproducción finalice. Esta opción no se puede combinar con REPEAT.
FULL SCREEN: solo para ficheros de video. Reproducir a pantalla completa.
WINDOW *tamaño x, tamaño y*: solo para ficheros de video. Reproducir en ventana del tamaño indicado.
LOCATE *x, y*: solo para ficheros de video. Reproducir en ventana, posicionándola en las coordenadas indicadas.
TITLE *cadena título*: solo para ficheros de video. Establece el título de la ventana de reproducción.

Para ficheros de audio, la reproducción se realiza sin mostrarse ventana, por lo que, las opciones FULL SCREEN, WINDOW, LOCATE y TITLE son ignoradas.

Para ficheros de video, por defecto, la reproducción se realiza en una ventana escalada al 100% del tamaño del video.

Por defecto, la reproducción se efectúa en segundo plano, asíncronamente, por lo que el programa continúa ejecutándose a la vez que se produce la reproducción.

Usando la orden GOTO, se pueden realizar saltos en la reproducción, a instantes de tiempo determinados.

Usando la función CREC se obtiene la posición actual de la reproducción (en milisegundos).

EOF indica si la reproducción acabó.

Ejemplos:

```
OPEN MEDIA #1, “tema1.mp3”
PLAY #1
WHILE NOT EOF(#1)
   LOCATE 10,1:PRINT CREC(#1)
WEND
CLOSE #1
```

```
OPEN MEDIA #1, “video1.avi”
PLAY #1, LOCATE 200, 100, TITLE “Mi video”, WAIT
CLOSE #1
```

POP

Tipo: Orden de programa

Sintaxis:

POP
POP *variable*
POP *variable*, *variable, …*

POP #*número pila*
POP #*número pila, variable*
POP #*número pila, variable*, *variable, …*

Descripción:

Desapila uno o más datos de una de las pilas del usuario, y opcionalmente los guarda en variables.

POP puede usarse con o sin especificación de número de pila. En caso de hacerlo, éste debe ser un número entre 0 y 31.

Si no es indica número de pila, se asume la cero, que se considera como pila principal.

POP sin argumentos desapila el dato de la cima de la pila correspondiente, ignorando su contenido.

Usando POP con uno o más argumentos, éstos deben de ser nombres de variables, numéricas o de cadena, cuyo tipo debe de coincidir con datos presentes en la pila.

El orden de despilado es el de aparición de los variables en POP.

Hay que tener en cuenta que, para recuperar contenidos de variables apiladas por PUSH, deben usarse POPs en orden inverso.

Ejemplos:

PUSH 10, 20, “un texto”, 30
PUSH #1, 100
POP → elimina el valor 30
POP a$, b, c
POP #1, d
PRINT a$, b, c → Imprime: un texto 20 10 100

PRINT

Tipo: Orden de programa

Sintaxis:

PRINT *dato*
PRINT *dato, dato …*
PRINT *dato; dato …*

PRINT #*número fichero, …*

Descripción:

Escribe uno o más datos en pantalla o en un fichero de texto o de red.

PRINT tiene distintas sintaxis, siendo la más genérica:

PRINT *dato*

… que imprime el dato en pantalla. El "dato" puede ser tanto una expresión numérica como de cadena.

Sin embargo, bajo una misma orden PRINT, pueden escribirse más de un dato en la misma línea:

PRINT *dato1*, *dato2*; *dato3*

… que imprime 3 datos, realizando una tabulación de separación entre *dato1* y *dato2*, y sin separación entre *dato2* y *dato3* (excepto si son números).

El uso de los caracteres "," y ";" dentro de la orden, sirven para separar los distintos datos a escribir. Usando la ",", además se indica que se debe realizar una tabulación. Por otra parte, ";" no realiza separación alguna.

Como excepción, los datos numéricos siempre incorporan un espacio delantero y trasero, aún con el uso del carácter ";".

Adicionalmente, puede añadirse un ";" al final de la orden PRINT, lo cual indica que no se realice el salto de línea, de manera que posteriores órdenes PRINT escribirán a continuación, en la misma línea.

PRINT también se usa para escribir en ficheros de textos y en ficheros de red (usualmente TCP), realizándose la especificación del número de fichero. La sintaxis es igual que el PRINT de pantalla.

Ejemplos:

num = 100
PRINT num → escribe 100
PRINT num, 200, "texto" → escribe 100 200 texto
PRINT num; 200; "texto" → escribe 100 200 texto
PRINT "texto";
PRINT "2" → escribe texto2

```
OPEN TEXT CREATE #1, “datos.txt”
PRINT #1, 100, 200, “texto”
CLOSE #1
```

PRINT USING

Tipo: Orden de programa

Sintaxis:

PRINT USING *cadena formato; expresión numérica …*

PRINT #*número fichero,* USING …

Descripción:

Escribe una expresión numérica usando un formato determinado.

PRINT USING es igual al PRINT regular, excepto en que la primera expresión numérica se escribe de acuerdo al formato especificado en “*cadena formato*”.

“*cadena formato*” representa el formato que debe adoptar el número, esto es, la cantidad de cifras enteras y decimales que debe tener, y opcionalmente, caracteres extras añadidos.

Las cifras del formato se representan por # ó 0:

= la posición puede contener una cifra del número, o bien un espacio en blanco.
0 = la posición puede contener una cifra del número, o bien un 0.

Usualmente, los caracteres extras a insertar serán los separadores de millares.

PRINT USING también se puede utilizar para ficheros de texto y de red.

Ejemplos:

PRINT USING “###”; 12 → imprime 12
PRINT USING “000”; 12 → imprime 012
PRINT USING “###.##”; 12 → imprime 12.
PRINT USING “###.0#”; 12 → imprime 12.0
PRINT USING “#,###.00”; 1200.2 → imprime 1,200.20
PRINT USING “###”;12;”texto” → imprime 12texto

PROCEDURE

Tipo: Orden de programa

Sintaxis:

PROCEDURE *nombre procedimiento*
PROCEDURE *nombre procedimiento*(IN *variable,* INOUT *variable,* OUT …)

PROCEDURE *nombre procedimiento … RETURNS variable*

Descripción:

Declara un procedimiento de usuario.

PROCEDURE se usa para declarar un bloque constructivo PROCEDURE-ENDPROC que forma un procedimiento con su propio ámbito de ejecución.

Un procedimiento está definido por un nombre único, una lista opcional de argumentos, de entrada, salida o entrada/salida y un valor de retorno también opcional (RETURNS).

Los valores de los parámetros que recibirá el procedimiento (cuando sea llamado) son mapeados en las variables indicadas en la declaración, que son locales al procedimiento.

Un procedimiento es invocado con la orden CALL (o con las funciones CALL y CALL$), y termina cuando encuentra ENDPROC.

Dentro de un procedimiento, todas sus variables son locales, no teniendo acceso a variables de otros procedimientos.

El acceso a variables globales es posible mediante uso del prefijo "__" (doble subrayado).

Se permiten procedimientos recursivos (se llaman a sí mismos con CALL).

Las 32 pilas de usuarios son globales y son accesibles desde cualquier procedimiento.

Los argumentos del procedimiento pueden ser numéricos, de cadena, tanto escalares como matrices, ya sean de entrada o de salida:

IN *variable*: declara un argumento de entrada escalar, cuyo valor estará en la variable indicada.

IN *variable()*: declara un argumento de entrada matricial.

OUT *variable*: declara un argumento de salida escalar. El valor de la variable será copiado a la variable indicada en CALL.

OUT *variable()*: declara un argumento de salida matricial.

INOUT *variable*: declara un argumento de entrada y salida escalar.

INOUT *variable()*: declara un argumento de entrada y salida matricial.

La diferencia entre OUT y INOUT está en que, un parámetro OUT, al comienzo del procedimiento, la variable asociada del procedimiento no tendrá valor, porque ésta es solo de salida, a diferencia de INOUT, que sí tendrá valor de entrada y además, en la finalización, será copiado a la variable asociada en CALL.

De forma opcional puede especificarse un valor de retorno, usando la clausula RETURNS, e indicando a continuación el nombre de la variable local del procedimiento que se usará para dar dicho valor de retorno. El tipo del valor de retorno es el que corresponda a la variable que se indique.

Los procedimientos con valores de retorno son útiles cuando estos sean llamados por medio de las funciones CALL y CALL$.

Ejemplos:

```
REM Variables locales y globales
a = 1 : b = 2 : c = 100
call sumar(a, b, c)
print a; b; c   → imprime 1  2  3
end
procedure sumar(in c, in d, out e)
   print a,b;c;d;e  → imprime 0  0  1  2  0  (con el call anterior)
   e = a+b
endproc   → al finalizar, copia el contenido de "e" en "c" del call anterior
procedure sumar2(in c, in d, inout e)
   print a,b;c;d;e  → imprime 0  0  1  2  100  (con el call anterior)
   e = a+b
endproc   → al finalizar, copia el contenido de "e" en "c" del call anterior

REM Inversión caracteres de una cadena
call invertir("abcdefg", inv$)
call invertir2("abcdefg", inv2$)
print inv$, inv$2          → imprime "gfedcba     gfedcba"
end
' Version iterativa
procedure invertir(in a$, out b$)
   for i = len(a$) to 1 step -1
      inc b$, a$[i]
   next
endproc
' Version recursiva
procedure invertir2(in a$, out b$)
   if not empty(a$) then call invertir(left$(a$, len(a$)-1), c$):b$=right$(a$, 1)+c$
endproc
' Versión recursiva con valor de retorno
procedure invertir3(in a$) returns b$
```

```
    if len(a$) > 1 then b$ = right$(a$, 1) + call$(invertir3(a$[1 .. #len(a$)-1])) \
                    else b$ = a$
endproc

REM Suma los elementos de un vector
OPTION BASE 1
DIM a(5)
for i = 1 to 5:a(i)=1:next
call sumar(a, 5, b)
print b  → imprime 15
end
procedure sumar(in vector(), in num, out suma)
   for i = 1 to num
      inc suma, vector(i)
   next
endproc

REM Llamada a procedimiento sin parámetros
print "Pulsa una tecla para acabar"
call esperar_tecla
end
procedure esperar_tecla
   while empty(inkey$):pause 100:wend
endproc
```

PUSH

Tipo: Orden de programa

Sintaxis:

PUSH *dato*
PUSH *dato*, *dato, …*

PUSH #*número pila*, *dato*
PUSH #*número pila*, *dato*, *dato, …*

Descripción:

Apila uno o más datos en una de las pilas de usuario.

PUSH puede usarse con o sin especificación de número de pila. En caso de hacerlo, éste debe ser un número entre 0 y 31.

Si no es indica número de pila, se asume la cero, que se considera como pila principal.

PUSH apila los datos indicados, sean numéricos o cadenas, en la pila correspondiente de usuario, en el orden indicado.

Posteriormente podrán recuperarse los datos usando POP.

Una de las utilidades de usar una pila de usuario es salvaguardar el contenido de variables.

Ejemplos:

a = 1 : b = 2
PUSH a, b
PRINT a, b → Imprime 1 2
a = 100 : b = 200
PRINT a, b → Imprime 100 200
POP b, a → se recuperan las variables en orden inverso
PRINT a, b → Imprime 1 2

PUT

Tipo: Orden de programa

Sintaxis:

PUT #*numero fichero*
PUT #*numero fichero, expresión número registro*

Descripción:

Escribe un registro de un fichero binario estructurado:

PUT #*numero fichero*: escribe el siguiente registro del fichero
PUT #*numero fichero, expresión número registro*: escribe el registro indicado en "expresión número registro".

Antes de realizar PUT, hay que asignar valores a los campos del fichero, usando las funciones FIELD y FIELD$.

Al realizarse el PUT, el puntero del fichero se sitúa sobre el siguiente registro.

Ejemplos:

OPEN BINARY #1, "datos.dat", FIELDS num=INT(4)
FIELD(#1, num) = 100
PUT #1, 10 → escribe el registro 10
FIELD(#1, num) = 200
PUT #1 → escribe el registro 11

RANDOMIZE

Tipo: Orden de programa

Sintaxis:

RANDOMIZE

Descripción:

Inicializa el generador de números aleatorios.

Normalmente, se utilizará RANDOMIZE al principio del programa, si éste utiliza las funciones RND o MXRND.

Ejemplos:

```
RANDOMIZE
PRINT RND
```

READ

Tipo: Orden de programa

Sintaxis:

READ *variable*
READ *variable, variable, …*

READ #*número fichero*, *variable cadena, longitud*
READ #*número fichero*, *variable cadena, longitud, timeout*

Descripción:

READ tiene 2 interpretaciones:

READ *variable, …*: lee uno o más datos que están contenidos en sentencias DATA del ámbito de ejecución actual. La lectura se realiza en dónde se encuentre el puntero interno de lectura, y se almacena el dato en la variable indicada. El dato y la variable deben de coincidir en su tipo. Al realizarse la lectura de un dato, el puntero se mueve convenientemente.

READ #*número fichero*, *variable cadena, longitud*: para ficheros binarios raw y de red, lee un número de bytes (indicado en "*longitud"*), y los guarda en la variable de cadena. Opcionalmente, se puede indicar un tiempo de espera máximo, o timeout, especificado en milisegundos (solo ficheros de red).

Fichero binarios raw:

La especificación de timeout es ignorada.

La lectura se inicia desde la posición actual.

Puede ocurrir que se lean menos datos que los indicados en "longitud", esto ocurre si se alcanza el fin de fichero. En tal situación, la lectura es correcta, no generarse error.

Ejemplos:

READ n → lee el siguiente dato de DATA (debe ser un número)
READ m, a$ → lee los dos siguientes datos de DATA (número y cadena)

OPEN NETWORK UDP(1001) #1, "192.168.1.101"
READ #1, a$, 100, 1500 → lee 100 bytes con un timeout de 1.5 segundos

OPEN BINARY #1, "datos.dat"
READ #1, A$, 20 → lee primeros 20 bytes
READ #1, B$, 5 → lee siguientes 5 bytes
GOTO #1, 40
READ #1, C$, 1 → lee byte número 40
READ #1, d$, 512 → lee bloque de 512 bytes
IF LEN(d$) < 512 THEN PRINT "fin de fichero"

REM

Tipo: Orden de programa

Sintaxis:

REM *comentario*

Descripción:

Inserta un comentario en el programa.

Esta orden no tiene ninguna acción asociada, y cuando se encuentra, todo el resto de la línea se ignora, y se pasa a ejecutar la siguiente línea.

Su utilidad es la de poder insertar comentarios dentro del programa.

Otra forma de añadir comentarios es mediante uso del carácter "'".

Ejemplos:

REM Ahora se imprimirá el número 100
PRINT 100
' este es otro comentario

RENUM

Tipo: Orden inmediata de edición

Sintaxis:

RENUM
RENUM *número primera línea*
RENUM *número primera línea, incremento*

Descripción:

Renumera las líneas del programa actual.

Para programas sin números de línea, RENUM realiza una renumeración de las líneas virtuales.

RENUM siempre actúa sobre todo el programa, y dispone de 3 sintaxis:

RENUM: renumera usando el número 10 como primera línea, utilizando un incremento de 10.
RENUM *número primera línea*: renumera usando como primera línea la indicada. Se utiliza un incremento de 10.
RENUM *número primera línea, incremento*: renumera usando como primera línea e incremento, los especificados.

RENUM se utiliza para reorganizar el programa, y para permitir insertar nuevas líneas entre medio de otras existentes.

Ejemplos:

RENUM 100, 5 → renumera dando a la primera línea el número 100. Incremento de 5 en 5.

REPEAT

Tipo: Orden de programa

Sintaxis:

REPEAT

Descripción:

Comienza un bucle REPEAT-UNTIL.

El bucle, al menos, se ejecutará una vez, hasta llegar a UNTIL, en donde se encuentra la condición de salida.

El bucle se ejecutará hasta que la condición indicada en UNTIL se cumpla.

Ejemplos:

```
i = 0
REPEAT
   PRINT i
   INC i
UNTIL i = 5
```

RESTORE

Tipo: Orden de programa

Sintaxis:

RESTORE
RESTORE *número línea*

Descripción:

Establece el puntero interno de lectura READ-DATA.

RESTORE permite mover el puntero interno a cualquier posición del programa.

Si no se especifica número de línea, el puntero se situará sobre la primera orden DATA existente.

RESTORE, al igual que DATA y READ, está limitado al ámbito de ejecución actual.

Ejemplos:

```
10 READ n → lee 100
20 RESTORE 70
30 READ m → lee 300
40 END
50 DATA 100
60 DATA 200
70 DATA 300
```

RETURN

Tipo: Orden de programa

Sintaxis:

RETURN

Descripción:

Retorna de una subrutina.

RETURN funciona junto a GOSUB, de manera que, cuando GOSUB realiza la llamada a la subrutina indicada, cuando, dentro de ella, se encuentre un RETURN se devolverá el control a la siguiente orden existente después del GOSUB.

Todo RETURN debe de estar equilibrado con todo GOSUB.

Se permite realizar RETURN dentro de bucles.

Ejemplos:

```
10 GOSUB 100
20 PRINT "despues de subrutina"
.
.
100 PRINT "inicio de subrutina"
110 RETURN
```

RIGHT$

Tipo: Función de cadena

Sintaxis:

RIGHT$(*cadena, número caracteres*)

Descripción:

Devuelve una subcadena formada por los últimos "*número caracteres*" de la cadena indicada.

Ejemplos:

PRINTRIGHT$("primero", 4) → imprime "mero"

RMDIR

Tipo: Orden de programa

Sintaxis:

RMDIR *cadena*

Descripción:

Borra el directorio indicado en la cadena de caracteres.

Ejemplos:

RMDIR "c:\temp"

RMFILE

Tipo: Orden de programa

Sintaxis:

RMFILE *cadena nombre fichero*

Descripción:

Borra el fichero indicado en la cadena de caracteres.

Ejemplos:

RMFILE “datos.dat”

RND

Tipo: Función numérica

Sintaxis:

RND
RND(*expresión numérica*)

Descripción:

Devuelve un número decimal aleatorio en el intervalo [0, 1] ó [0, 1).

Usando RND ó RND(1) se devuelve un número en [0, 1], mientras que con RND(0) se devuelve en el intervalo [0, 1)

Ejemplos:

PRINT INT(RND * 10) → imprime un número entero aleatorio entre 0 y 10
PRINT INT(RND(0) * 10) → imprime un número entero aleatorio entre 0 y 9

ROL

Tipo: Función numérica

Sintaxis:

ROL(*expresión numérica, desplazamiento*)

Descripción:

Devuelve el desplazamiento a la izquierda indicado de la expresión numérica.

La expresión regular se interpreta como número entero.

Ejemplos:

PRINT ROL(100, 2) → imprime 400

ROR

Tipo: Función numérica

Sintaxis:

ROR(*expresión numérica, desplazamiento*)

Descripción:

Devuelve el desplazamiento a la derecha indicado de la expresión numérica.

La expresión regular se interpreta como número entero.

Ejemplos:

PRINT ROR(100, 2) → imprime 25

ROUND

Tipo: Función numérica

Sintaxis:

ROUND(*expresión numérica*)
ROUND(*expresión numérica, número decimales*)

Descripción:

Calcula y devuelve el redondeo (por exceso o defecto) de la expresión numérica.

Opcionalmente, puede indicarse un número de decimales, de forma que el número es redondeado a esa cantidad de decimales.

En caso de no indicarse, el número es redondeado sin decimales.

Ejemplos:

PRINT ROUND(100.2) → imprime 100
PRINT ROUND(100.5) → imprime 101
PRINT ROUND(100.523, 2) → imprime 100.52

RUN

Tipo: Orden de programa

Sintaxis:

RUN
RUN *cadena nombre fichero*

Descripción:

Ejecuta el programa actual o bien uno especificado.

Típicamente RUN se usará en modo interactivo para poner en funcionamiento el programa.

En caso de utilizar RUN sin argumentos, se ejecuta el programa actual.

RUN puede usarse también con una cadena como argumento, haciendo que el programa indicando con la misma se ejecute (borrando el actual).

Todas las variables son puestas a cero.

RUN SUBPROGRAM

Tipo: Orden de programa

Sintaxis:

RUN SUBPROGRAM *cadena nombre fichero*, *nombre subprograma*
RUN SUBPROGRAM *nombre subprograma*

Descripción:

Ejecuta un subprograma, opcionalmente cargándolo de forma previa.

En caso de estar ya cargado el subprograma, éste se puede ejecutar directamente con:

RUN SUBPROGRAM *nombre subprograma*

Típicamente RUN SUBPROGRAM se usará en subprogramas que sean "interfaces" a librerías DLL externas, esto es, que tengan declaraciones EXTERN PROCEDURE, de forma que puedan ser invocables desde fuera.

La ejecución del subprograma se realiza dentro del entorno del programa actual.

Una vez finalizado, el programa actual proseguirá.

Ejemplos:

```
‘ Ejecuta el subprogram interfaz a Windows
RUN SUBPROGRAM “win.bas”, win
CALL Sleep@win(1000)
```

SAVE

Tipo: Orden de programa

Sintaxis:

SAVE *nombre fichero*
SAVE *nombre fichero*, A
SAVE *nombre fichero*, B
SAVE *nombre fichero*, P

Descripción:

Guarda el programa actual en fichero.

Hay varios formatos de almacenamiento del programa:

SAVE *nombre fichero*: guarda el programa en formato texto usando el juego de caracteres Windows ANSI. Externamente el fichero puede editarse.
SAVE *nombre fichero*, A: guarda el programa en formato texto usando el juego de caracteres ASCII/OEM. Externamente el fichero puede editarse.
SAVE *nombre fichero*, B: guarda el programa en formato binario. Externamente el fichero no puede editarse.
SAVE *nombre fichero*, P: igual al anterior, pero protegiendo el programa, de manera, que al cargarlo de nuevo (con LOAD) no permitirá efectuar ninguna orden de edición, tal como LIST, EDIT o DELETE.

El nombre de fichero puede tener o no extensión. En caso de no especificarla, se añade .BAS.

Ejemplos:

SAVE “programa1.bas”
SAVE “programa2”, P

SCREEN FULL MODE

Tipo: Orden de programa

Sintaxis:

SCREEN FULL MODE

Descripción:

Establece la ejecución del programa a pantalla completa.

SCREEN WINDOW MODE

Tipo: Orden de programa

Sintaxis:

SCREEN WINDOW MODE *NumFilas, NumColumnas*

Descripción:

Establece la ejecución del programa a una ventana de las dimensiones indicadas.

Las dimensiones se dan en caracteres de pantalla.

Esta orden se puede usar también para cambiar el tamaño actual de la ventana de ejecución.

Ejemplos:

SCREEN WINDOW MODE 40, 80 → Ejecución a ventana de 80 columnas y 40 filas.

SCROLL

Tipo: Orden de programa

Sintaxis:

SCROLL *fila1, columna1, fila2, columna2* TO fila*dest, columnadest*

Descripción:

Realiza un movimiento (scroll) de una región de la pantalla.

Esta orden mueve la zona rectangular de la pantalla definida por las coordenadas (fila1, columna1, fila2, columna2) a la posición (filadest, columnadest), siendo (fila1, columna1) la esquina superior izquierda a mover, (fila2, columna2) la esquina inferior derecha y (filadest, columnadest) la esquina superior izquierda destino.

Como consecuencia del movimiento, se quedará una zona de pantalla vacia, la cual se rellenará con espacios en blanco usando el color actual.

SCROLL se usará principalmente para realizar movimientos verticales u horizontales.

Ejemplos:

SCROLL 10, 1, 15, 20 TO 9, 1 → realiza un desplazamiento vertical hacia arriba

SCRCHR$

Tipo: Función de cadena

Sintaxis:

SCRCHR$(*fila1, columna1, fila2, columna2*)

Descripción:

Devuelve los caracteres de una región de la pantalla.

SCRCHR$ devuelve una cadena con los caracteres presentes en la zona de la pantalla especificada.

La cadena devuelta es lineal, estando los caracteres de una línea concatenados con los de la línea anterior.

Ejemplos:

LOCATE 10, 1 : PRINT "El árbol de la colina es grande"

A$ = SCRCHR$(10, 16, 10, 21) → A$ = "colina"

SEARCH
SEARCHE
SEARCHI
SEARCHN
SEARCHNI

Tipo: Función numérica

Sintaxis:

SEARCH(*array, dato a buscar*)
SEARCH(*array(desde índice..hasta índice), dato a buscar*)

SEARCH(*arraycadena* [*desde carácter..hasta carácter*]*, cadena*)
SEARCH(*arraycadena*(*desde índ..hasta índ*)[*desde carac..hasta carac*]*, cadena*)
SEARCHE(*arraycadena* …
SEARCHI(*arraycadena* …
SEARCHN(*arraycadena* …
SEARCHNI(*arraycadena* …

Descripción:

Busca un dato (numérico o cadena) en un array.

SEARCH devuelve el índice de la primera ocurrencia del dato de la dimensión a recorrer del array indicado.

En caso de no encontrarse el valor, devuelve -1.

Para arrays de más de una dimensión, deberá indicarse una especificación de dimensiones de forma que queden fijas todas las dimensiones menos una, que será en donde se realizará la búsqueda.

La búsqueda puede hacerse sobre todo el array, o bien, sobre un rango de elementos, indicado por la especificación “desde índice..hasta índice”.

En caso de arrays de cadenas, se puede especificar desde y hasta qué índices de caracteres (indicados entre corchetes) se debe de usar para realizar la búsqueda. Si no se indican, ésta se realizará considerando toda la cadena completa.

SEARCHE, SEARCHI, SEARCHN y SEARCHNI son funciones alternativas disponibles solo para arrays de tipo cadena, cuyo comportamiento es distinto en función de si deben realizar una búsqueda exacta y de considerar mayúsculas/minúsculas:

Función	Búsqueda exacta	Sensible a mayúsculas
SEARCH	Si	Si
SEARCHE	Si	Si
SEARCHI	Si	No
SEARCHN	No	Si
SEARCHNI	No	No

Las búsquedas no-exactas (SEARCHN, SEARCHNI) consideran la cadena a buscar como una subcadena que debe aparecer al comienzo de alguna de las cadenas del array.

El array no es necesario que esté ordenado.

Ejemplos:

```
DIM a(3)
a(0)=10:a(1)=20:a(2)=30:a(3)=20
PRINT SEARCH(a, 15) → imprime -1
PRINT SEARCH(a, 20) → imprime 1
PRINT SEARCH(a(2..3), 20) → imprime 3

DIM a(2, 2)
a(0, 0)=10:a(1, 0)=20:a(2, 0)=30:a(3, 0)=20
a(0, 1)=10:a(0, 2)=20
PRINT SEARCH(a, 20) → error
PRINT SEARCH(a(*,  0), 20) → imprime 1
PRINT SEARCH(a(*,  1), 20) → imprime -1
PRINT SEARCH(a(1..2,  2), 20) → imprime 2

DIM b$(3)
b$(0)=”Madrid”:b$(1)=”Toledo”:b$(2)=”Avila”:b$(3)=”Segovia”
```

PRINT SEARCH(b$, “Toledo”) → imprime 1
PRINT SEARCH(b$, “segovia”) → imprime -1
PRINT SEARCHI(b$, “segovia”) → imprime 3
PRINT SEARCH(b$, “Ma”) → imprime -1
PRINT SEARCHN(b$, “Ma”) → imprime 0
PRINT SEARCH(b$, “le”) → imprime -1
PRINT SEARCH(b$[3..6], “le”) → imprime -1
PRINT SEARCHN(b$[3..6], “le”) → imprime 1
PRINT SEARCHN(b$[3..6], “LE”) → imprime -1
PRINT SEARCHNI(b$[3..6], “LE”) → imprime 1

SELECT-CASE-ENDSEL

Tipo: Orden de programa

Sintaxis:

SELECT *expresión selección*
CASE *lista expresiones caso 1*
.
.
CASE *lista expresiones caso 2*
.
.
ELSE
.
.
ENDSEL

Descripción:

Ejecuta una serie de instrucciones en función del valor de un selector.

SELECT es una estructura de bloque que implementa un condicional múltiple, y puede usarse tanto para expresiones numéricas como de cadena.

Su equivalente en bloques IF-THEN-ELSE sería:

IF *expresión selección* IN [*lista expresión caso 1*] THEN
.
.
ELSE
IF *expresión selección* IN [*lista expresión caso 2*] THEN
.
.
ELSE

.
.
ENDIF

En ejecución, cuando se encuentra un SELECT, se ejecutarán aquellas instrucciones del CASE cuya "*lista expresión caso*" contenga el valor de la expresión indicada en SELECT. Si ningún CASE coincide, se ejecutarán las instrucciones del ELSE, en caso de haberlas.

Una vez ejecutadas las opciones del CASE (o ELSE) coincidente, el resto de alternativas son ignoradas, y el programa continua después del ENDSEL.

En caso de existir varios CASE cuyas expresiones se solapen, se ejecutará el coincidente que primero se encuentre.

Ejemplos:

```
INPUT "Número opción: ";opcion
SELECT opcion
  CASE  1
    PRINT "Has elegido la opción 1"
  CASE 2, 3
    PRINT "Has elegido la opción 2 ó 3"
  CASE 4..10, 12
    PRINT "Has elegido la opción 4 al 10 ó la 12"
  ELSE
    PRINT "Has elegido otra opción"
ENDSEL
```

```
INPUT "Introduce un dia de la semana: ";dia$
SELECT LOWER$(dia$)
  CASE  "lunes"
    PRINT "Has elegido el lunes"
  CASE "martes", "miercoles"
    PRINT "Has elegido el martes o el miércoles"
  CASE ""
    PRINT "No has elegido ningún día"
  ELSE
    PRINT "Has elegido otro día de la seman"
ENDSEL
```

SET ANGLE

Tipo: Orden de programa

Sintaxis:

SET ANGLE RAD
SET ANGLE DEG
SET ANGLE GRAD

Descripción:

Establece la unidad angular a usar en las funciones trigonométricas.

Se permiten tres tipos de sistemas:

SET ANGLE RAD: radianes (0 .. 2π)
SET ANGLE DEG: grados sexagesimales (0 .. 360^{o})
SET ANGLE GRAD: grados centesimales (gradianes) (0 .. 400^{g})

Por defecto, se utilizan radianes.

Ejemplos:

PRINT ASIN(1) → imprime 1.57079633 (PI/2) (radianes)

SET ANGLE DEG
PRINT ASIN(1) → imprime 90 (grados)

SET ANGLE GRAD
PRINT ASIN(1) → imprime 100 (gradianes)

SET ENDIAN

Tipo: Orden de programa

Sintaxis:

SET ENDIAN LITTLE
SET ENDIAN BIG

Descripción:

Establece el modo de representación interna de números enteros, a "big endian" o "little endian".

El modo de representación es tenido en cuenta en:

- Funciones CV* y MK*$
- Ficheros binarios (campos de números enteros)
- Ficheros de red (comunicaciones TCP y UDP)

El modo "big endian" se usará en ficheros de red, ya que, de forma estándar, la representación en TCP/IP debe estar en este formato.

El modo es compartido para todos los subprogramas cargados.

Por defecto, el modo es "Little endian".

Ejemplos:

a$ = MKI$(100) → representación del entero 100 en Little Endian
SET ENDIAN BIG
PRINT CVI(a$) → imprime 25600
SET ENDIAN LITTLE
PRINT CVI(a$) → imprime 100

SET MODE

Tipo: Orden de programa

Sintaxis:

SET MODE MATRIX
SET MODE SCALAR

Descripción:

Establece el modo de ejecución del programa, matricial o escalar.

Usando el modo matricial se activan las capacidades de cálculo matricial de iBASIC y se posibilita el uso de las funciones matriciales (que comienzan por MX).

SET MODE se pueden usar tantas veces sea necesario.

Por defecto, el modo de ejecución es escalar.

Ejemplos:

SET MODE MATRIX
PRINT MXDET(a) →imprime el determinante de la matriz "a"
SET MODE SCALAR

SGN

Tipo: Función numérica

Sintaxis:

SGN(e*xpresión numérica*)

Descripción:

Devuelve el signo de la expresión numérica:

-1 : expresión es negativa
0 : expresión es cero
1 : expresión es positiva

Ejemplos:

PRINT SGN(-0.5) → imprime -1

SIN

Tipo: Función numérica

Sintaxis:

SIN(*expresión numérica*)

Descripción:

Calcula y devuelve el seno de la expresión numérica indicada.

Ejemplos:

y = SIN(x)

SORT
SORTI

Tipo: Orden de programa

Sintaxis:

SORT *array*
SORT *array(desde índice..hasta índice)*

SORT *array cadena*[*desde carácter..hasta carácter*]
SORT *array cadena*(desde *índice..hasta índice*)[*desde carácter..hasta carácter*]

SORTI *array cadena* …

Descripción:

Ordena un array (numérico o de cadena) en orden ascendente.

SORT realiza la ordenación, usando el algoritmo Quicksort, de un array completo, o bien, de una parte de él en caso de especificar un rango de índices (desde, hasta).

En caso de arrays de cadenas, se puede especificar desde y hasta qué índices de caracteres (indicados entre corchetes) se debe de usar para realizar la ordenación. Si no se indican, la ordenación se efectuará sobre toda la longitud de la cadena.

SORTI es sólo válido para arrays de cadenas, y efectúa una ordenación no sensible a mayúsculas/minúsculas.

Ejemplos:

DIM a(3)
a(0)=100:a(1)=20:a(2)=30:a(3)=20
SORT a(0..1)
PRINT a(0), a(1) → imprime 20 100
SORT a
PRINT a(0), a(1) → imprime 20 20

DIM nombre$(4)
nombre$(1) = “Luis”
nombre$(2) = “pablo”
nombre$(3) = “Juan”
nombre$(4)=”Pedro”
SORT nombre$
FOR i = 1 to 4:PRINT nombre$(i):NEXT → Juan Luis Pedro **pablo**
SORTI nombre$
FOR i = 1 to 4:PRINT nombre$(i):NEXT → Juan Luis **pablo** Pedro
SORT nombre$[2..3]
FOR i = 1 to 4:PRINT nombre$(i):NEXT → p**ab**lo P**ed**ro J**ua**n L**ui**s

SOUND

Tipo: Orden de programa

Sintaxis:

SOUND *frecuencia, duración*

Descripción:

Produce un sonido de la frecuencia y duración indicadas.

La frecuencia debe estar en el rango 32 a 32767 Hz.

La duración se especifica en milisegundos.

El sonido será generado por el altavoz del ordenador.

Ejemplos:

SOUND 440, 500 → produce una nota “La”

SPACE$

Tipo: Función de cadena

Sintaxis:

SPACE$(*número espacios*)

Descripción:

Devuelve una cadena con el número de espacios indicado.

Ejemplos:

a$=”primero”+SPACE$(10)+”segundo”
PRINT LEN(a$) → imprime 24

SQRT

Tipo: Función numérica

Sintaxis:

SQRT(*expresión numérica*)

Descripción:

Calcula y devuelve la raíz cuadrada de la expresión numérica indicada.

Ejemplos:

y = SQRT(x)

SREC

Tipo: Función numérica de fichero

Sintaxis:

SREC(*#numero fichero*)

Descripción:

Devuelve el tamaño de registro del fichero indicado.

Dependiendo del tipo de fichero, la interpretación de SREC es distinta:

Ficheros de texto: devuelve siempre 0.

Ficheros binarios raw: devuelve siempre 1.

Ficheros binarios estructurados: devuelve el tamaño del registro definido por FIELDS en la orden OPEN.

Ficheros multimedia: devuelve siempre 1.

Ficheros de red: devuelve el número de bytes recibidos pero que están pendientes de ser leídos.

Para el caso particular de ficheros de red, SREC es de gran utilidad, debido a que se puede usar de forma previa a realizar una lectura con READ, y así determinar el número de bytes recibidos.

Ejemplos:

```
OPEN BINARY #1, "datos.dat", FIELDS num=INT(4), num2=FLOAT(8)
PRINT SREC(#1) → imprime 12
```

```
OPEN NETWORK UDP(1000) #1, "192.168.1.100"
a$= MKI$(1)
WRITE #1, a$
PAUSE 1500
READ #1, b$, SREC(#1)
```

STOP

Tipo: Orden de programa

Sintaxis:

```
STOP
STOP #numero fichero
```

Descripción:

Detiene la ejecución de programa o la reproducción de un fichero multimedia.

STOP tiene dos interpretaciones en función de su sintaxis:

STOP: detiene la ejecución del programa.
STOP *#numero fichero*: detiene la reproducción del fichero multimedia indicado. La reproducción no se podrá reanudar, aunque sí, comenzar de nuevo con PLAY.

STR$

Tipo: Función de cadena

Sintaxis:

STR$(*expresión numérica*)

Descripción:

Devuelve una cadena con el valor de la expresión numérica convertido a texto ASCII.

Ejemplos:

num=100
PRINT STR$(num), LEN(STR$(num)) → imprime 100 3

STRING$

Tipo: Función de cadena

Sintaxis:

STRING$(*expresión numérica*, *subcadena*)

Descripción:

Devuelve una cadena que está formada por la concatenación de la subcadena indicada un número de veces dado por el valor de la expresión numérica.

Ejemplos:

PRINT STRING$(10, "*") → imprime **********
PRINT STRING$(2, "repetido") → imprime repetidorepetido

SUM

Tipo: Función numérica

Sintaxis:

SUM(*expresión numérica*, *expresión numérica, expresión numérica …*)

Descripción:

Calcula y devuelve la suma de la lista de expresiones numéricas indicadas (separadas por comas).

En modo matricial, devuelve la suma de las matrices.

Ejemplos:

PRINT SUM(2, 100, 3*20) → imprime 162

SWAP

Tipo: Orden de programa

Sintaxis:

SWAP *variable1, variable2*

Descripción:

Intercambia los contenidos de las dos variables indicadas.

El tipo de las variables ser el mismo.

En modo matricial, se permite especificar submatrices, siempre que sean del mismo tamaño.

Ejemplos:

```
num1 = 10:num2 = 20
SWAP num1, num2
PRINT num1, num2 → imprime 20  10
```

```
SET MODE MATRIX
DIM c
c = MXRND(4, 4)
SWAP c(1, *), c(2, *) → intercambia las filas 1 y 2
```

TAN

Tipo: Función numérica

Sintaxis:

TAN(*expresión numérica*)

Descripción:

Calcula y devuelve la tangente de la expresión numérica indicada.

Ejemplos:

```
y = TAN(x)
```

TIME

Tipo: Variable numérica del sistema

Sintaxis:

TIME

Descripción:

Devuelve el instante de tiempo actual, en milisegundos, desde que el sistema está en funcionamiento.

Típicamente, TIME se usará para generar marcas de tiempo (*timestamps*) y también para calcular tiempos de ejecución de fragmentos de código.

Ejemplos:

```
t1 = TIME
FOR i = 1 TO 100
  .
```

.
NEXT
t2 = TIME
PRINT “Tiempo de proceso:“; (t2-t1)/1000;”segundos”

TIME$

Tipo: Variable de cadena del sistema

Sintaxis:

TIME$

Descripción:

Devuelve una cadena con la hora actual del sistema, con precisión de milisegundos.

La formato de la cadena es: HH:MM:SS.mmm

Ejemplos:

PRINT TIME$ → 12:01:30.128 (por ejemplo)

TRIM$

Tipo: Función de cadena

Sintaxis:

TRIM$(*cadena*)

Descripción:

Devuelve la cadena indicada eliminando delimitadores y considerando el carácter ASCII 0.

La cadena devuelta consiste en la eliminación de los espacios y tabuladores delanteros y traseros.

En caso de que la cadena contenga el carácter ASCII 0, ésta es truncada esta dicha posición, de forma previa a la eliminación de los separadores.

TRIM$ será de utilidad en llamadas a procedimientos externos que usan cadenas ASCIIZ.

Ejemplos:

PRINT “*”;TRIM$(“ Roma “);”*” → imprime *Roma*

```
A$ = “abc”+CHR$(0)+”def”
B$ = TRIM$(A$)
PRINT B$, LEN(B$)  → imprime abc    3
```

TRUE

Tipo: Constante del sistema

Sintaxis:

```
TRUE
```

Descripción:

Devuelve el valor correspondiente a condición verdadera, que es uno.

El contrario de TRUE es FALSE.

Ejemplos:

```
WHILE TRUE
   PRINT “Esto es un bucle infinito”
WEND
```

UNLOAD SUBPROGRAM

Tipo: Orden de programa

Sintaxis:

UNLOAD SUBPROGRAM *nombre subprograma*

Descripción:

Descarga un subprograma previamente cargado.

El subprograma a descargar debe haber sido cargado previamente con LOAD SUBPROGRAM o RUN SUBPROGRAM, y además, no debe estar en ejecución.

Una vez descargado, no se podrá acceder a sus contenidos, excepto si se vuelve a cargar.

Ejemplos:

```
LOAD SUBPROGRAM “utiles.bas”, util
CALL convertir_numero@util(100, c$)
‘ Descargar
UNLOAD SUBPROGRAM util
```

UNTIL

Tipo: Orden de programa

Sintaxis:

UNTIL *expresión condición*

Descripción:

Finaliza y define la condición de salida de un bucle REPEAT-UNTIL.

El bucle, al menos, se ejecutará una vez, hasta llegar a UNTIL, en donde se encuentra la condición de salida.

El bucle se ejecutará hasta que la condición indicada en UNTIL se cumpla.

Ejemplos:

```
i = 0
REPEAT
   PRINT i
   INC i
UNTIL i = 5
```

UPPER$

Tipo: Función de cadena

Sintaxis:

UPPER$(*cadena*)

Descripción:

Devuelve la conversión a mayúsculas de la cadena indicada.

UPPER$ es la simétrica a LOWER$.

Ejemplos:

PRINT UPPER$("Rio grande") → imprime "RIO GRANDE"

USING$

Tipo: Función de cadena

Sintaxis:

USING$(*cadena formato, expresión numérica*)

Descripción:

Devuelve una cadena con la aplicación de la expresión numérica indicada sobre el formato especificado.

USING$ devuelve lo mismo que PRINT USING imprimiría en pantalla.

Véase PRINT USING para detalles.

Ejemplos:

PRINT USING$(“0000.00”, 120.2) → Imprime 0120.20

VAL

Tipo: Función numérica

Sintaxis:

VAL(*cadena*)

Descripción:

Devuelve el valor numérico de la cadena.

La cadena debe contener un número para que la función tenga sentido. En caso contrario, devuelve 0.

VAL es la función simétrica de STR$.

Ejemplos:

num$=”100”
PRINT VAL(num$)*2 → imprime 200

VOLUME

Tipo: Variable numérica del sistema

Sintaxis:

VOLUME
VOLUME *expresión numérica*

Descripción:

Devuelve o establece el factor del volumen de la salida de audio.

Usando VOLUME sin expresión numérica, devuelve un número decimal entre 0 y 1

Usando VOLUME con una expresión numérica, se establece el volumen. El valor de la expresión debe estar entre 0 y 1.

Para ambos casos, el valor que se devuelve o establece, es un número entre 0 y 1 es un factor del volumen máximo de la salida de audio.

Un valor de 0 indica silenciamiento.

Un valor de 1 indica volumen máximo.

Ejemplos:

VOLUME 0.5 → establece el volumen al 50% del máximo
PRINT VOLUME → imprime 0.5

WHILE
WEND

Tipo: Orden de programa

Sintaxis:

WHILE *expresión condición*
  …
WEND

Descripción:

WHILE-WEND conforma un bucle, cuyo cuerpo de órdenes se repetirá mientras la condición indicada en WHILE se cumpla (sea verdadera).

La condición se comprueba en cada iteración, y puede ocurrir que su cuerpo no se ejecute.

Ejemplos:

```
i = 0
WHILE i < 5
   PRINT i
   INC i
WEND
```

WINDOW TITLE

Tipo: Orden de programa

Sintaxis:

WINDOW TITLE *cadena*

Descripción:

Establece el título de la ventana de ejecución del programa actual.

Ejemplos:

WINDOW TITLE “Mi programa”

WRITE

Tipo: Orden de programa

Sintaxis:

WRITE #*número fichero*, *cadena*

Descripción:

Envia o escribe los bytes, contenidos en la cadena, al fichero indicado (solo ficheros binarios raw o de red).

Ficheros binarios raw:

El contenido de la cadena es escrito completamente en el fichero, a partir de la posición actual.

Para añadir al final del fichero, debe usar previamente la orden GOTO:

GOTO #*número fichero*, NREC(#*número fichero*) + 1

Ficheros de red:

WRITE sólo se aplica para ficheros de red (TCP o UDP), y envía todo el contenido de la cadena indicada al host y puerto que se especificó en OPEN.

Dependiendo de los protocolos de aplicación, la cadena a enviar puede verse como buffer que contiene el mensaje o trama a enviar, y que se formará mediante el uso de las funciones MK*$.

Ejemplos:

OPEN NETWORK UDP(1001) #1, "192.168.1.101"
WRITE #1, MKI$(1) → envía el entero 1 al host

OPEN BINARY #1, "datos.dat"
READ #1, a$, 2 → lee los dos primeros bytes
GOTO #1, 1
WRITE #1, "90" → escribe 2 bytes (90) al principio del fichero

XPOS

Tipo: Variable numérica del sistema

Sintaxis:

XPOS

Descripción:

Devuelve la posición coordenada X actual del cursor en pantalla.

Ejemplos:

```
REM Imprime * en la fila 10 sin mover la columna
LOCATE 10, XPOS : PRINT "*"
```

XSIZE

Tipo: Variable numérica del sistema / Función numérica

Sintaxis:

```
XSIZE

XSIZE(#número fichero)
XSIZE(#0)
```

Descripción:

En general, XSIZE devuelve el tamaño de la dimensión X (o anchura) de una ventana o de la pantalla, dependiendo de la sintaxis usada:

XSIZE: devuelve el número total de columnas (valor máximo de X) de la pantalla o ventana de ejecución de iBASIC o del programa compilado.

XSIZE(#*número fichero*): para ficheros multimedia de video, devuelve la anchura en píxeles (valor máximo de X) de la ventana de reproducción.

XSIZE(#0): devuelve la anchura en píxeles de la pantalla física.

Ejemplos:

```
REM Imprime * en el centro de la fila 10
LOCATE 10, XPOS / 2 : PRINT "*"

OPEN MEDIA #1, "video1.avi"
PLAY #1
PRINT XSIZE(#1)
```

YPOS

Tipo: Variable numérica del sistema

Sintaxis:

```
YPOS
```

Descripción:

Devuelve la posición coordenada Y actual del cursor en pantalla.

Ejemplos:

```
REM Imprime * en la columna 30 sin mover la fila.
LOCATE YPOS, 30 : PRINT "*"
```

YSIZE

Tipo: Variable numérica del sistema / Función numérica

Sintaxis:

YSIZE

YSIZE(#*número fichero*)
YSIZE(#0)

Descripción:

En general, YSIZE devuelve el tamaño de la dimensión Y (o altura) de una ventana o de la pantalla, dependiendo de la sintaxis usada:

YSIZE: devuelve el número total de filas (valor máximo de Y) de la pantalla o ventana de ejecución de iBASIC o del programa compilado.

YSIZE(#*número fichero*): para ficheros multimedia de video, devuelve la altura en píxeles (valor máximo de Y) de la ventana de reproducción.

YSIZE(#0): devuelve la altura en píxeles de la pantalla física.

Ejemplos:

```
REM Imprime * en el centro de la pantalla
LOCATE YPOS / 2, XPOS / 2:PRINT "*"
```

Anexo 1, Códigos de error

Código	Descripción	Código	Descripción
0	Sin error	82	Fallo al borrar directorio
1	Error léxico, carácter no esperado	83	Fallo al borrar fichero
2	Error léxico, número incorrecto	84	Timeout no permitido
3	Error léxico, carácter no encontrado	85	Inject no permitido
4	Error léxico, símbolo demasiado largo	86	Fichero no compatible con GET
5	FOR sin NEXT	87	Fichero no compatible con PUT
6	NEXT sin FOR	88	Error de lectura desde red
7	WHILE sin WEND	89	Error de escritura en red
8	WEND sin WHILE	90	Fallo al asignar valor a variable
9	REPEAT sin UNTIL	91	Fallo al asignar valor a constante
10	UNTIL in REPEAT	92	Elementos de array no pueden ser constantes
11	Indice de bucle no coincide	93	Demasiadas dimensiones
12	BREAK fuera de bucle	94	Especificación inválida de dimensiones
13	CONTINUE fuera de bucle	95	Especificación de OPTION BASE incorrecta
14	IF sin ENDIF	96	OPTION BASE no permitido
15	ELSE sin IF/SELECT/CHAIN	97	Coordenadas fuera de rango
16	Bloque ELSE duplicado	98	Fallo al cambiar a modo pantalla completa
17	ENDIF sin IF	99	Fallo al cambiar a modo ventana
18	Bloque IF-ELSE-END inválido	100	Fallo al asignar título de ventana
19	BREAK IF fuera de bloque IF	101	Fallo al realizar scroll de pantalla
20	GOSUB sin RETURN	102	Fallo al realizar operación sobre fichero multimedia
21	RETURN sin GOSUB	103	Fallo al reproducir fichero multimedia
22	SELECT sin ENDSEL	104	Fallo al reanudar fichero multimedia
23	CASE sin SELECT	105	Fallo al detener fichero multimedia
24	ENDSEL sin SELECT	106	Fallo al pausar fichero multimedia
25	Bloque SELECT-ENDSEL inválido	107	Fallo al establecer volumen
26	CHAIN sin ENDCHAIN	108	Fallo al obtener dimensiones de reproducción
27	NODE sin CHAIN	109	Opciones de reproducción incorrectas
28	ENDCHAIN sin CHAIN	110	Etiqueta duplicada
29	Bloque CHAIN-ENDCHAIN inválido	111	Etiqueta no existe
30	Número no esperado	112	Sentencia con número de línea no permitido
31	Cadena no esperada	113	Programa ya tiene etiquetas
32	Palabra clave no esperada	114	Procedimiento duplicado
33	Variable de cadena no esperada	115	Procedimiento no existe
34	Variable numérica no esperada	116	Ejecución alcanzó un procedimiento
35	Definición de etiqueta no esperado	117	Argumento duplicado en procedimiento
36	Signo no esperado	118	Nombre de argumento no válido
37	Operador lógico no esperado	119	Faltan parámetros en llamada a

			procedimiento
38	Nueva sentencia no esperada	120	Demasiados parámetros
39	Carácter no esperado	121	Tipo incorrecto de parámetro
40	Especificación de intervalo no esperado	122	Fallo al llamar a procedimiento externo
41	Orden incompleta	123	ENDPROC sin llamada CALL a procedimiento
42	Falta especificación de un número	124	Procedimiento sin ENDPROC
43	Falta palabra clave	125	Función duplicada
44	Falta identificador	126	Nombre no válido de función
45	Falta variable	127	Faltan parámetros en llamada a función
46	Falta especificación de intervalo	128	Subprograma no existe
47	Falta carácter	129	Falta nombre de subprograma
48	Falta palabra clave	130	Subprograma no permitido
49	Falta signo de comparación	131	Error en subprograma
50	Fichero no encontrado	132	Fallo al descargar subprograma
51	Fallo de apertura de fichero	133	Fallo al realizar PUSH
52	Fallo en lectura de fichero	134	Fallo al realizar POP
53	Fallo en escritura de fichero	135	Número de pila incorrecto
54	Programa con errores	136	Error de sintaxis
55	Programa con errores	137	Línea no existe
56	Programa con errores	138	Número de línea incorrecto
57	Programa con errores	139	División por cero
58	Nombre de fichero no válido	140	Memoria insuficiente
59	Fallo al grabar programa	141	Número inválido
60	Número de fichero incorrecto	142	Error leyendo de READ
61	Fichero ya abierto	143	Línea no es de DATA o no existe
62	Fichero no abierto	144	Número fuera de rango
63	Fichero no está abierto en modo texto	145	Error de numeración de líneas
64	Fichero no está abierto en modo binario (raw)	146	Fallo al eliminar números de línea
65	Fichero no está abierto en modo binario (estrcturado)	147	Array no permitido
66	Fichero no está abierto en modo multimedia	148	Símbolo no es un array
67	Fichero no está abierto en modo red	149	Símbolo no es un array de cadena
68	Fichero no está abierto en modo red TCP	150	Símbolo no es numérico
69	Fichero no está abierto en modo red UDP	151	Fallo al ejecutar comando
70	Fichero no está abierto en modo lectura	152	No se permite mezcla de sentencias con y sin números de línea
71	Fichero no está abierto en modo escritura	153	Orden solo permitida con números de línea
72	Fallo al realizar operación sobre fichero	154	Programa está protegido
73	Longitud de campo incorrecta	155	Fallo al ordenar array
74	Modo de fichero incorrecto	156	Fallo al buscar en array

75	Falta especificación de campos	157	Fallo al compilar
76	Tipo de campo no coincide	158	Error interno de ejecución
77	Campo no existe	1000	Matrices no son compatibles
78	Fallo al asignar valor a campo	1001	El resultado no es un escalar
79	Fallo al obtener valor de campo	1002	Término no es un escalar
80	Fallo al cambiar a directorio	1003	Función no disponible en modo escalar
81	Fallo al crear directorio		

Anexo 2, Códigos de color

Código	Color
0	Negro
1	Azul
2	Verde
3	Cian
4	Rojo
5	Fucsia
6	Marrón
7	Blanco
8	Gris
9	Azul brillante
10	Verde brillante
11	Cian brillante
12	Rojo brillante
13	Fucsia brillante
14	Amarillo brillante
15	Blanco brillante

Anexo 3, Códigos de teclas especiales

Tecla	Normal	Shift	Ctrl	Alt
F1	00 3B	00 54	00 5E	00 68
F2	00 3C	00 55	00 5F	00 69
F3	00 3D	00 56	00 60	00 6A
F4	00 3E	00 57	00 61	00 6B
F5	00 EF	00 58	00 62	00 6C
F6	00 40	00 59	00 63	00 6D
F7	00 41	00 5A	00 64	00 6E
F8	00 42	00 5B	00 65	00 6F
F9	00 43	00 5C	00 66	00 70
F10	00 44	00 5D	00 67	00 71
F11	E0 85	E0 87	E0 89	E0 8B
F12	E0 86	E0 88	E0 8A	E0 8C
Insert	E0 52	E0 52	E0 92	00 A2
Del (Suprimir)	E0 53	E0 53	E0 93	00 A3
Home (Inicio)	E0 47	E0 47	E0 77	00 97
End (Fin)	E0 4F	E0 4F	E0 75	00 9F
Page Up (RePag)	E0 49	E0 49	E0 86	00 99
Page Dn (AvPag)	E0 51	E0 51	E0 76	00 A1
Flecha arriba	E0 48	E0 48	E0 8D	00 98
Flecha abajo	E0 50	E0 50	E0 91	00 A0
Flecha izquierda	E0 4B	E0 4B	E0 73	00 9B
Flecha derecha	E0 4D	E0 4D	E0 74	00 9D

Anexo 4, Mapa de caracteres

El mapa de caracteres define la grafía de los códigos ASCII desde el 0 al 255.

La siguiente línea BASIC ilustra cómo imprimir todo el juego de caracteres:

```
FOR i = 0 TO 255 : PRINT CHR$(i) : NEXT
```

iBASIC no utiliza el sistema Unicode, sino que utiliza los llamados códigos de página.

Un código de página define un mapa de 256 caracteres (de 0 a 255).

Habitualmente los caracteres de 0 a 127 son estándar, y de 128 a 255 son extendidos y varían de una página de código a otra.

En general iBASIC utiliza el código de página que define el mapa de caracteres OEM, usado en aplicaciones de consola.

El código es distinto en función de la configuración local de Windows, pudiendo ser 437, 850, 852, además de otros.

Se puede determinar el código de página usando la función CODEPAGE de iBASIC:

```
PRINT CODEPAGE
```

Sin embargo, en las operaciones de carga y grabación de programas BASIC (órdenes LOAD y SAVE), por defecto se utiliza el código de página estándar de Windows, ANSI 1252.

De esta forma, al cargar se realiza una conversión desde la página ANSI a la OEM y al grabar se hace la contraria, de la OEM a la ANSI.

Esto es así para permitir la edición de programas BASIC con editores de Windows.

Opcionalmente, las órdenes LOAD y SAVE pueden ser utilizadas añadiendo el parámetro “,A” para indicar que ha de utilizarse el código de página OEM en lugar del ANSI:

```
LOAD “fichero”, A
SAVE “fichero”, A
```

Código de página 850:

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F
80	Ç 00C7	ü 00FC	é 00E9	â 00E2	ä 00E4	à 00E0	å 00E5	ç 00E7	ê 00EA	ë 00EB	è 00E8	ï 00EF	î 00EE	ì 00EC	Ä 00C4	Å 00C5
90	É 00C9	æ 00E6	Æ 00C6	ô 00F4	ö 00F6	ò 00F2	û 00FB	ù 00F9	ÿ 00FF	Ö 00D6	Ü 00DC	ø 00F8	£ 00A3	Ø 00D8	× 00D7	ƒ 0192
A0	á 00E1	í 00ED	ó 00F3	ú 00FA	ñ 00F1	Ñ 00D1	ª 00AA	º 00BA	¿ 00BF	® 00AE	¬ 00AC	½ 00BD	¼ 00BC	¡ 00A1	« 00AB	» 00BB
B0	░ 2591	▒ 2592	▓ 2593	| 2502	┤ 2524	Á 00C1	Â 00C2	À 00C0	© 00A9	╣ 2563	║ 2551	╗ 2557	╝ 255D	¢ 00A2	¥ 00A5	┐ 2510
C0	└ 2514	┴ 2534	┬ 252C	├ 251C	─ 2500	┼ 253C	ã 00E3	Ã 00C3	╚ 255A	╔ 2554	╩ 2569	╦ 2566	╠ 2560	═ 2550	╬ 256C	¤ 00A4
D0	ð 00F0	Ð 00D0	Ê 00CA	Ë 00CB	È 00C8	ı 0131	Í 00CD	Î 00CE	Ï 00CF	┘ 2518	┌ 250C	█ 2588	▄ 2584	¦ 00A6	Ì 00CC	▀ 2580
E0	Ó 00D3	ß 00DF	Ô 00D4	Ò 00D2	õ 00F5	Õ 00D5	µ 00B5	þ 00FE	Þ 00DE	Ú 00DA	Û 00DB	Ù 00D9	ý 00FD	Ý 00DD	¯ 00AF	´ 00B4
F0	00AD	± 00B1	‗ 2017	¾ 00BE	¶ 00B6	§ 00A7	÷ 00F7	¸ 00B8	° 00B0	¨ 00A8	· 00B7	¹ 00B9	³ 00B3	² 00B2	■ 25A0	NBSP 00A0

Código de página 1252: (ANSI)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F
80	€ 20AC		‚ 201A	ƒ 0192	„ 201E	… 2026	† 2020	‡ 2021	ˆ 02C6	‰ 2030	Š 0160	‹ 2039	Œ 0152		Ž 017D	
90		‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014	˜ 02DC	™ 2122	š 0161	› 203A	œ 0153		ž 017E	Ÿ 0178
A0	NBSP 00A0	¡ 00A1	¢ 00A2	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	- 00AD	® 00AE	¯ 00AF
B0	° 00B0	± 00B1	² 00B2	³ 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	¹ 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C0	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D0	Ð 00D0	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	Ý 00DD	Þ 00DE	ß 00DF
E0	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F0	ð 00F0	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ý 00FD	þ 00FE	ÿ 00FF