2.5. QBASIC y Glosario de Comandos Usados

2.5.1. HISTORIA DEL QBASIC

BASIC es una familia de lenguajes de programación. Fue originalmente ideado como una herramienta de enseñanza, se diseminó entre los microcomputadores caseras en la década de 1980, y sigue siendo popular hoy en día en muchos dialectos bastante distintos del original.

El nombre de BASIC, significa Beginners All-purpose Symbolic Instruction Code (Código de Instrucción Simbólico Todo Propósito para Principiantes), está ligado al nombre de un trabajo sin publicar del coinventor del lenguaje, Thomas Kurtz (el nombre no está relacionado con la serie de C.K. Ogden, “Basic English”). Algunos críticos han hecho humor con el nombre, atribuyéndole como significado Bill's Attempt to Seize Industry Control (el Intento de Bill de Tomar el Control de la Industria) en respuesta a las políticas de Microsoft con respecto a los intérpretes incluidos con las primeras computadores compatibles con IBM PC.

2.5.2. Antecedentes

Antes de mediados de la década de 1960, los computadores eran herramientas sumamente caras que eran utilizadas únicamente para propósitos especiales, ejecutando una sola "tarea" a la vez. Sin embargo, durante esa década, los precios comenzaron a bajar al punto que incluso las pequeñas empresas podían costearlas. La velocidad de las máquinas se incrementó al punto que a menudo quedaban ociosas porque no había suficientes tareas para ellas.

Los lenguajes de programación de aquellos tiempos estaban diseñados como las máquinas en las que corrían: para propósitos específicos como el procesamiento de fórmulas. Como las máquinas para una sola tarea eran caras, se consideraba que la velocidad de ejecución era la característica más importante de todas. En general, todas eran difíciles de utilizar, y aportaban poca estética.

Fue en aquellos tiempos que el concepto de sistema de Tiempo compartido comenzó a popularizarse. En uno de estos sistemas, el tiempo de procesamiento del computador principal se dividía, y a cada usuario se le otorgaba una pequeña porción en una secuencia. Las máquinas eran lo suficientemente rápidas como para engañar a la mayoría de usuarios, dándoles la ilusión de que disponían de una máquina entera solo para ellos.

En teoría la distribución del tiempo entre los usuarios redujo considerablemente el costo de la computación, ya que una sola máquina podía ser compartida, al menos en teoría, entre cientos de usuarios.

2.5.3. Nacimiento y primeros años

El lenguaje BASIC original fue inventado en 1964 por John George Kemeny (1926-1993) y Thomas Eugene Kurtz (1928-) en el Dartmouth College. En los años subsiguientes, mientras que otros dialectos de BASIC aparecían, el BASIC original de Kemeny y Kurtz era conocido como BASIC Dartmouth.

BASIC fue diseñado para permitir a los estudiantes escribir programas usando terminales de computador de tiempo compartido. BASIC estaba intencionado para facilitar los problemas de complejidad de los lenguajes anteriores, con un nuevo lenguaje diseñado específicamente para la clase de usuarios que los sistemas de tiempo compartido permitían: un usuario más sencillo, a quien no le interesaba tanto la velocidad, sino el hecho de ser capaz de usar la máquina. Los diseñadores del lenguaje también querían que permaneciera en el dominio público, lo que contribuyó a que se diseminara.

Los ocho principios de diseño de BASIC fueron:

1. Ser fácil de usar para los principiantes.
2. Ser un lenguaje de propósito general.
3. Permitir que los expertos añadieran características avanzadas, mientras que el lenguaje permanecía simple para los principiantes.
4. Ser interactivo.
5. Proveer mensajes de errores claros y amigables.
6. Responder rápido a los programas pequeños.
7. No requerir un conocimiento del hardware de la computadora.
8. Proteger al usuario del sistema operativo.

El lenguaje fue en parte basado en FORTRAN II y en parte en Algol 60, con adiciones para hacerlo apropiado para tiempo compartido y aritmética de matrices, BASIC fue implementado por primera vez en la mainframe GE-265[3], que soportaba múltiples terminales. Contrario a la creencia popular, era un lenguaje compilado al momento de su introducción. Casi inmediatamente después de su lanzamiento, los profesionales de computación comenzaron a alegar que BASIC era muy lento y simple. Tal argumento es un tema recurrente en la industria de los computadores.

Aún así, BASIC se expandió hacia muchas máquinas, y se popularizó moderadamente en los minicomputadores como la serie DEC PDP y la Data General Nova. En estos casos, el

lenguaje era implementado como un intérprete, en vez de un compilador, o alternativamente, de ambas formas.

2.5.4. Crecimiento explosivo

Sin embargo, fue con la introducción de la Microcomputador Altair 8800 en 1975 que BASIC se diseminó ampliamente. La mayoría de lenguajes de programación eran demasiado grandes para caber en la pequeña memoria que la mayoría de usuarios podía pagar para sus máquinas, y con el lento almacenamiento que era la cinta de papel, y más tarde la cinta de audiocasete (los discos magnéticos aún no existían), y la falta de editores de texto adecuados, un lenguaje pequeño como BASIC era una buena opción. Uno de los primeros en aparecer fue Tiny BASIC, una implementación simple de BASIC escrita originalmente por el Dr. Li-Chen Wang, y portada más tarde a la Altair por Dennis Allison, a petición de Bob Albrecht (quien después fundó el *Dr. Dobbs Journal*). El diseño de Tiny BASIC y el código fuente completo fue publicado en 1976 en DDJ.

En 1977 Microsoft (entonces consistía de dos personas: Gates y Allen) lanzó Altair BASIC. Luego comenzaron a aparecer bajo licencia versiones para otras plataformas, y millones de copias y variantes pronto estarían en uso. Se convirtió en uno de los lenguajes estándar en la Apple II. Para 1979 Microsoft estaba negociando con varios vendedores de microcomputadores, incluyendo a IBM, para licenciar un intérprete de BASIC para sus computadores. Una versión se incluyó en los chips ROM de las PCs IBM, para PCs sin discos, y en las que disponían de unidad de diskettes el BASIC era iniciado automáticamente si es que no se colocaba ningún de diskette de arranque de sistema operativo.

Mientras que las nuevas compañías intentaban seguir los pasos del éxito de Altair, IMSAI, North Star, y Apple, creando la revolución de la computador casera. BASIC se convirtió en una característica estándar para casi todas las computadoras caseras; la mayoría venía con un intérprete de BASIC en ROM (algo hecho por primera vez por la Commodore PET en 1977). Pronto había muchos millones de computadores corriendo BASIC alrededor del mundo, un número mucho más grande que el de todos los usuarios de otros lenguajes juntos. Muchos programas, especialmente los de la Apple II e IBM PC, dependían de la presencia del intérprete de BASIC de Microsoft y no correrían sin éste; por lo que Microsoft usó la licencia de copyright en los intérpretes de BASIC para influenciar en las negociaciones con los vendedores de computadores.

El BASIC fue también el lenguaje prefijado en los computadores caseros europeos de la década de los 80 como el ZX Spectrum, MSX o el Commodore 64, haciendo muchas veces la función de intérprete y sistema operativo primitivo ya que venían implementados en ROM.

La suerte de BASIC dio un giro nuevamente con la introducción de Visual Basic de Microsoft. Aunque es algo difícil considerar este lenguaje como BASIC (a pesar de que usa muchas palabras clave conocidas de BASIC) se ha convertido en uno de los lenguajes más usados en la plataforma Windows, y se dice que representa del 70 al 80% del desarrollo comercial de aplicaciones. Visual Basic for Applications (VBA) fue añadido a Microsoft Excel 5.0 en 1993 y al resto de la línea de productos de Microsoft Office en 1997. Windows 98 incluyó un intérprete de VBScript. La versión más reciente de Visual Basic es llamada VB.NET. La suite OpenOffice.org incluye una variante de BASIC menos poderosa que su contraparte de Microsoft.

Una sintaxis muy mínima de BASIC sólo necesita los comandos LET, PRINT, IF y GOTO. Un intérprete que ejecuta programas con esta sintaxis mínima no necesita una pila. Algunas de las primeras implementaciones eran así de simples. Si se le agrega una pila, se pueden agregar también ciclos FOR tiertors anidados y el comando GOSUB. Un intérprete de BASIC con estas características necesita que el código tenga números de línea.

Los dialectos modernos de BASIC MIUN han abandonado los números de línea, y la mayoría (o todos) han añadido control de flujo estructurado y los constructores de declaración de datos como los de otros lenguajes como C y Pascal:

- do
- loop
- while
- until
- exit
- on ... goto
- gosub
- switch
- case

Variantes recientes como Visual Basic han introducido características orientadas a objetos, y hasta herencia en la última versión. La administración de memoria es más fácil que con muchos otros lenguajes de programación procedimentales debido al Recolector de basura (a costas de la velocidad de ejecución).

2.5.5. Procedimientos y Control de Flujo

BASIC no tiene una biblioteca externa estándar como otros lenguajes como C. En cambio, el intérprete (o compilador) contiene una biblioteca incorporada de procedimientos

intrínsecos. Estos procedimientos incluyen la mayoría de herramientas que un programador necesita para aprender a programar y escribir aplicaciones sencillas, así como funciones para realizar cálculos matemáticos, manejar cadenas, entrada de la consola, gráficos y manipulación de archivos.

Algunos dialectos de BASIC no permiten que los programadores escriban sus propios procedimientos. Los programadores en cambio deben escribir sus programas con un gran número de enunciados GOTO para hacer las ramificaciones del programa. Esto puede producir un código fuente muy confusa, comúnmente referido como Código espagueti. Los enunciados GOSUB ramifican el programa a una especie de subrutinas sin parámetros o variables locales. La mayoría de versiones de BASIC modernas, como Microsoft QuickBASIC han añadido soporte completo para subrutinas, funciones y programación estructurada. Este es otra área donde BASIC difiere de otros muchos lenguajes de programación. BASIC, como Pascal, hace una distinción entre un procedimiento que no devuelve un valor (llamado subrutina) y un procedimiento que lo hace (llamado función). Muchos otros lenguajes (como C) no hacen esa distinción y consideran todo como una función (algunas que devuelven un valor “void” [vacío]).

Mientras que las funciones que devuelven un valor son una adición relativamente reciente a los dialectos de BASIC, muchos de los primeros sistemas soportaban la definición de funciones matemáticas en línea, con DEF FN (“DEFine FunctioN” [DEFinir FunciónN]). El Dartmouth BASIC original también soportaba funciones al estilo de Algol, así como subrutinas desde sus primeros tiempos.

2.5.6. Tipos de Datos

BASIC es reconocido por tener muy buenas funciones para manipular cadenas. Los primeros dialectos ya tenían un juego de funciones fundamentales (LEFT$, MID$, RIGHT$) para manipular cadenas fácilmente. Como las cadenas son utilizadas en aplicaciones diarias, esta era una ventaja considerable sobre otros lenguajes al momento de su introducción.

El Dartmouth BASIC original soportaba únicamente datos de tipo numérico y cadenas. No había un tipo entero. Todas las variables numéricas eran de punto flotante. Las cadenas eran de tamaño dinámico. Los arreglos de ambos, números y cadenas, eran soportados, así como matrices (arreglos de dos dimensiones).

Cada dialecto moderno de BASIC posee al menos los tipos de datos entero y cadena. Los tipos de datos son generalmente distinguidos por un posfijo, los identificadores de cadenas terminan con $ (signo de dólar), mientras que los enteros no lo hacen. En algunos dialectos, las variables deben ser declaradas (con DIM) antes de usarse; otros dialectos no

lo requieren, pero tienen la opción para hacerlo (típicamente usando una directiva como OPTION EXPLICIT). Muchos dialectos también soportan tipos adicionales, como enteros de 16 y 32 bits y números de punto flotante. Adicionalmente algunos permiten la utilización de tipos definidos por el usuario, similar a los "records" de Pascal, o las "structs" de C.

La mayoría de dialectos de BASIC además de los arreglos de tipos primitivos, también soportan arreglos otros tipos. En algunos, los arreglos deben ser declarados antes de que puedan usarse (con el enunciado DIM). Es común también el soporte para arreglos de dos y más dimensiones.

```
DIM miArregloDeEnteros (100) AS INTEGER
DIM miListaDeNombres (50) AS STRING.
```

2.6. Glosario de ordenes o Statements

CLS

Borra la pantalla

SINTAXIS DEL COMANDO

CLS [{0 | 1 | 2}]

COMENTARIO

CLS Borra la ventana gráfica o la de texto. Si se ha establecido una ventana gráfica (usando VIEW), se borrará sólo la ventana de texto o toda la pantalla.

CLS 0 Borra la pantalla, quitando todo el texto y los gráficos.

CLS 1 Borra la ventana de gráficos, o la pantalla completa si no se ha establecido una ventana para gráficos

CLS 2 Borra la ventana de texto

DECLARE

Declara una función o un sub programa

SINTAXIS DEL COMANDO

DECLARE {FUNCTION | SUB} nombre [([lista parámetros])]

COMENTARIO

- nombre — El nombre del procedimiento
- lista parámetros — Una o más variables que especifican los parámetros que serán pasados al procedimiento cuando éste sea llamado

  variable [( )] [AS tipo] [, variable [( )] [AS tipo]]...

  variableEl nombre de una variable Basic.

  tipo — El tipo de datos de la variable (INTEGER, LONG, SINGLE, DOUBLE, STRING o un tipo de datos definido por el usuario).
- DECLARE es requerido si se hace un llamado a un procedimiento SUB sin CALL. Q Basic generará automáticamente instrucciones DECLARE cuando se guarde un programa.

DIM

DIM.- Declara un arsenal o especifica un tipo de datos para una variable nonarray.

REDIM.- Declara o vuelve a clasificar según el tamaño un arsenal dinámico, borrando cualquier valor anterior.

SINTAXIS DEL COMANDO

DIM [SHARED] variable[(subscripts)] [AS type] [,variable[(subscripts)] [AS type]]

REDIM [SHARED] variable(subscripts) [AS type][,variable(subscripts) [AS type]]

COMENTARIO

* SHARED.- Especifica que las variables están compartidas con todo el SUB or FUNCTION los procedimientos de la FUNCTION en el módulo.
* Variable.- El nombre de un arsenal o de una variable.
* Subscript.- Dimensiones del arsenal, expresadas en lo siguiente: [lower TO] upper [,[lower TO]
* Coger.- El límite más bajo de los subíndices de arsenal. El límite más bajo del defecto es cero.
* Upper.- El límite superior
* AS type.- Declara el tipo de datos del arsenal o del variable (INTEGER, LONG, SINGLE, DOUBLE, STRING, o de un tipo de datos definido por el usario).
* DIM.- Declara las órdenes estáticas o dinámicas.

CLOSE

Cierra uno o más archivos o dispositivos abiertos.

SINTAXIS DEL COMANDO

CLOSE [[#]numarchivo%[, [#] numarchivo%]...]

COMENTARIO

- numarchivo% El número de un archivo o dispositivo abierto
- CLOSE sin argumentos cerrará todos los archivos y dispositivos abiertos.

END

Pone fin a un programa, procedimiento, bloque o tipo de datos definido por el usuario.

SINTAXIS DEL COMANDO

END [{DEF | FUNCTION | SELECT | SUB | TYPE}]

COMENTARIO

- DEF Termina la definición de una función DEF FIN que ocupe varias líneas
- FUNCTION Termina la definición de un procedimiento FUNCTION
- IF Termina un bloque de instrucciones IF...THEN...ELSE
- SELECT Termina un bloque SELECT CASE
- SUB Termina un procedimiento SUB
- TYPE Termina la definición de un tipo de datos definido por el usuario
- Si no se especifica ningún argumento, END pondrá fin al programa y cerrará todos los archivos

FOR... NEXT

Repite un bloque de instrucciones el número de veces especificado

SINTAXIS DEL COMANDO

FOR contador = inicio TO fin [STEP incremento [bloqueinstrucciones]

NEXT [contador [, contador]...]

COMENTARIO

- contador Una variable numérica utilizada como contador de bucle
- inicio y fin Las variables inicial y final del contador
- incremento El incremento con el que se cambia el contador cada vez que se ejecute el bloque

GET y PUT

GET lee lo que se encuentra en un archivo en un almacenador intermediario o una variable de acceso aleatorio.

PUT escribe un almacenador intermediario variable o de acceso aleatorio a un archivo.

SINTAXIS DEL COMANDO

GET [#]filenumber%[,[recordnumber&][,variable]]

PUT [#]filenumber%[,[recordnumber&][,variable]]

COMENTARIO

Filenumber.- Número de un archivo abierto.

Recornumber.- Sirve para los archivos de acceso aleatorio, el número del expediente a leer o a escribir. Para el binario-modo archivo, donde la posición del octeto esta leyendo o escribiendo desde el comienzo.

Variable.- Para GET, una variable usada para recibir la entrada del archivo. Para PUT, una variable que contiene salida para escribir al archivo. La variable es generalmente una variable de un tipo de datos definido por el usuario.

IF ... THEN ... ELSE

Ejecuta una instrucción o bloque de instrucciones según las condiciones especificadas.

SINTAXIS DEL COMANDO

IF condición 1 THEN

[bloqueinstrucciones-1]

[ELSEIF condición2 THEN

[bloqueinstrucciones-2]]...

[ELSE

[bloqueinstrucciones-n]

END IF

COMENTARIO

IF condición THEN instrucciones [ELSE instrucciones]

- condición1 — Cualquier expresión que puede ser evaluada
  condición2 — Como verdadera (nocero) o falsa (cero)
- bloque-instrucciones1
  bloque-instrucciones2
  bloque-instrucciones-n

KILL

Elimina archivos del disco

SINTAXIS DEL COMANDO

KILL archivo$

COMENTARIO

- archivo$ — Identifica el archivo o los archivos que serán eliminados
  Puede incluir una ruta de acceso y las condiciones ? y
  * DOS

LEN

Devuelve el número de caracteres en una cadena, o el número de bytes requeridos para almacenar una variable.

SINTAXIS DEL COMANDO

LEN(expresión-cadena$)

LEN(variable)

COMENTARIO

- expresión – cadena — Cualquier expresión de cadena
- variable — Cualquier variable que no sea de cadena

INPUT y LINE INPUT

Lee la información desde el teclado o desde un archivo. LINE INPUT lee una línea de hasta 255 caracteres desde el teclado o desde un archivo.

SINTAXIS DEL COMANDO

INPUT [;] [“mensaje” {; | ,}] lista variables

LINE INPUT [;] [“mensaje”;] variable$

INPUT #numarchivo%, listavariables

LINE INPUT #numarchivo%, variable$

COMENTARIO

- mensaje — Una cadena literal optativa que será presentada antes de que el usuario introduzca datos. Un punto y coma después del mensaje agregará un signo de interrogación al texto del mensaje.

- listavariables — Una o más variables, separadas con comas, en las que serán almacenados los datos introducidos desde el teclado o leídos desde un archivo. Los nombres de variables pueden tener hasta 40 caracteres y deben comenzar con una letra. Los caracteres válidos son A – Z, 0 – 9 y el punto (.). No se pueden usar letras acentuadas ni la ñ.
- variable$ — Almacena una línea de caracteres introducidos desde el teclado o leídos desde un archivo
- numarchivo% — El número de un archivo abierto
- INPUT utiliza la coma separador de entradas.
  LINE INPUT lee todos los caracteres hasta encontrar un retorno de carro.
- Para datos introducidos desde el teclado, un punto y coma inmediatamente después de INPUT mantendrá el cursor en la misma línea después que el usuario presione la tecla ENTER.

LOOP

Repite un bloque de instrucciones mientras una condición tenga el estado verdadero, o hasta que una condición adquiera el estado verdadero

SINTAXIS DEL COMANDO

DO WHILE [{WHILE | UNTIL}, condicion]
[bloqueinstrucciones]
LOOP
DO
[bloqueinstrucciones]
LOOP [{WHILE | UNTIL}, condicion]

COMENTARIO

- condición — Una expresión numérica que Basic evalúa como verdadero (un cero) o falso (cero).

OPEN

Abre un archivo o dispositivo

SINTAXIS DEL COMANDO

OPEN archivo$ [FOR modo] [ACCESS acceso] [bloqueo] AS [#]numarch% [LEN=longreg

COMENTARIO

■ Archivo$; El nombre del archivo o dispositivo. El nombre puede incluir una unidad de disco y ruta de acceso.

■ Modo; Uno de los siguientes modos de archivo: APPEND, BINARY INPUT, OUTPUT o RANDOM. Vea ◄Modos de archivo para OPEN►.

■ Acceso; En una red, especifica si el archivo será abierto con el tipo de acceso READ, WRITE o READ WRITE. Vea ◄Claúsula ACCESS en instrucción OPEN►.

■ Bloqueo; Especifica el estado de bloqueo de archivos en una red: SHARED, LOCK READ, LOCK WRITE, LOCK READ WRITE.

■ Numarch%; Un número entre 1 y 255 que identifica el archivo mientras está abierto.

■ Longreg%; Para archivos de acceso aleatorio, la longitud de registro (el valor predeterminado es 128 bytes). Para archivos secuenciales, el número de caracteres en búfer (el valor predeterminado es 512 bytes).

REM

Permite que las observaciones explicativas sean insertadas en un programa.

SINTAXIS DEL COMANDO

REM remark

COMENTARIO

* Remark Cualquier texto.
* Remarks se no hacen caso cuando el programa funciona a menos que contengan metacommands.
* Una observación se puede insertar en una línea después de una declaración ejecutable si él es precedido por solo-cotizan (') la forma de REM o si se precede el REM por dos puntos (:).

SELECT CASE

Ejecuta uno de varios bloques de la declaración dependiendo del valor de una expresión.

SINTAXIS DEL COMANDO

```
SELECT CASE testexpression
CASE expressionlist1
[Statementblock-1]
[CASE expressionlist2
[Statementblock-2]]...
[CASE ELSE
[Statementblock-n]]
END SELECT
```

COMENTARIO

Testexpression cualquier expresión numérica o de la secuencia.

expressionlist1 una o más expresiones para emparejar el testexpression.

expressionlist2 Esta palabra clave debe preceder de cualquier operador emparentado en una expresión.

Statementblock-1 unas o más declaraciones sobre unas o más líneas.

Statementblock-2

Statementblock-n

Las discusiones del expressionlist pueden tener cualquiera de estas formas o de a ¦ combinación de ellas, separada por comas:

expression[,expression]...

expression TO expression

IS relational-operator expression

Expresión numérica o de la secuencia compatible ¦ con el testexpression. ¦

Relational-operator uno de los operadores emparentados siguientes: ¦ <, < =, >, > =, < >, o =.

SUB

Define un procedimiento SUB.

SINTAXIS DEL COMANDO

SUB nombre [(lista parámetros)] [STATIC]

[bloque instrucciones]

END SUB

COMENTARIO

- nombre — El nombre de un procedimiento SUB, de hasta 40 caracteres, sin sufijo indicando el tipo de datos.
- lista parámetros — Una o más variables que especifican los parámetros que serán pasados a un procedimiento SUB cuando éste sea llamado

  variable [( )] [AS tipo] [, variable [( )] [AS tipo]]...

  variableEl nombre de una variable Basic.

  Tipo — El tipo de datos de la variable (INTEGER, LONG, SINGLE, DOUBLE, STRING o un tipo de datos definido por el usuario).
- STATIC — Especifica que los valores de las variables locales del procedimiento SUB sean guardados entre llamados a la función.

- Cuando hace el llamado a un procedimiento SUB, podrá especificar que el valor de un argumento no sea cambiado por el procedimiento, poniendo el argumento entre paréntesis.

TYPE

Contiene un tipo de datos que contiene uno o más elementos

SINTAXIS DEL COMANDO

TYPE tipousuario

Elemento AS tipo

[Elemento AS tipo]

END TYPE

COMENTARIO

- tipo usuario — El nombre del tipo de datos que será definido. El nombre puede tener hasta cuarenta caracteres y debe comenzar con una letra. Los caracteres válidos son A – Z, 0 – 9 y el punto (.). No se pueden usar letras acentuadas ni la ñ.
- elemento — Un elemento del tipo de datos del elemento (INTEGER, LONG, SINGLE, DOUBLE, STRING o un tipo de datos definido por el usuario)
- Use DIM, REDIM, COMMON, STATIC o SHARED para crear una variable con un tipo de datos definido por el usuario.

UCASE$

Convierte cadenas en letras minúsculas o letras mayúsculas

SINTAXIS DEL COMANDO

LCASE$ (expresión-cadena$)

UCASE$ (expresión-cadena$)

COMENTARIO

- expresión-cadena$ — Cualquier expresión de cadena

FUNCTION

Define un procedimiento FUNCTION

SINTAXIS DEL COMANDO

FUNCTION nombre [(lista parámetros)] [STATIC]

[bloqueinstrucciones]

nombre=expresión

[bloqueinstrucciones]

END FUNCTION

COMENTARIO

- nombre — El nombre de la función y tipo de datos que devuelve especificado por un sufijo de tipo de datos (%, &, !, #, o $).
- lista parámetros — Una o más variables que especifican los parámetros que serán pasados a la función cuando ésta sea llamado

  variable [( )] [AS tipo((, variable (( )((AS tipo((...

  variable — El nombre de una variable Basic.

tipo El tipo de datos de la variable (INTEGER, LONG, SINGLE, DOUBLE, STRING o un tipo de datos definido por el usuario).

- STATIC Especifica que los valores de las variables locales de la función serán guardados entre llamados a la función.
- Expresión El valor de la función devuelto
- Cuando llama una función, podrá especificar que el valor de un argumento no sea cambiado por la función, poniendo el argumento entre paréntesis

RANDOM

Las palabras clave APPEND, BINARY, INPUT, OUTPUT y RANDOM se utilizan en la instrucción OPEN para especificar modos de archivo. INPUT, OUTPUT y RANDOM también se utilizan en la instrucción OPEN COM.

COMENTARIO

- APPEND especifica que el archivo será abierto para dar información de salida secuencial y coloca el puntero de archivo al final del archivo. Una instrucción PRINT # o WRITE # luego anexa información al archivo.
- BINARY especifica el modo de archivo binario. En este modo, es posible leer o escribir información en cualquier posición de byte del archivo usando instrucciones GET o PUT.
- INPUT especifica que el archivo será abierto para recibir información de entrada secuencial.

- OUTPUT especifica que el archivo será abierto para dar información de salida secuencial.

- RANDOM especifica que el archivo será abierto en el modo de acceso aleatorio. RANDOM es el modo de archivo predeterminado.

CALL

Transfiere el control a un procedimiento SUB.

SINTAXIS DEL COMANDO

[CALL] nombre [([listargumentos])]

COMENTARIO

■ Nombre.- El nombre del procedimiento SUB que será llamado.

■ Listargumentos.- Las variables o constantes que serán pasados al procedimiento SUB. Separe los argumentos con comas. Para especificar argumentos de matrices, use el nombre la matriz seguida de paréntesis vacíos.

■ Si omite la palabra clave CALL, también deberá omitir los paréntesis alrededor de listargumentos. Declare el procedimiento en una instrucción. DECLARE antes de llamarlo, o guarde el programa y QBasic generará auto-máticamente una instrucción DECLARE.

■ Para especificar un argumento cuyo valor no será cambiado por el procedimiento, encierre el argumento entre paréntesis.

COLOR

Establece los colores presentados en la pantalla.

SINTAXIS DEL COMANDO

COLOR [primerplano%] [,[fondo%] [,bordes%]] Modo de pantalla 0 (sólo texto)

COLOR [fondo%] [,paleta%] Modo de pantalla 1

COLOR [primerplano%] Modos de pantalla 4, 12, 13

COLOR [primerplano%] [,fondo&] Modos de pantalla 7-10

COMENTARIO

■ Primerplano%; Un número que establece el color de primer plano de primerplano&, la pantalla. En el modo de pantalla 0, primerplano% es un atributo de color que establece el color al texto. En otros modos de pantalla, primerplano% es un atributo de color o valor de color de 4 bytes (modo de pantalla 4 solamente) que establece el color para el texto y líneas dibujadas.

■ Fondo%; Un número que establece el color de fondo en la fondo&, pantalla. En el modo de pantalla 0, fondo% es un atributo de color. En el modo de pantalla 1, fondo% es un valor de color de 4 bits. En los modos de pantalla 7-10, fondo& es un valor de color.

■ Bordes%; Un atributo de color que establece el color de los bordes de la pantalla.

■ Paleta%; Un número (0 ó 1) que especifica el juego de atributos de color que será utilizado:

Paleta%	Atributo 1	Atributo 2	Atributo 3
0	Verde	Rojo	Marrón
1	Azul-verdoso	Magenta	Blanco Intenso

■ Los atributos y valores de color disponibles dependerán del adaptador de gráficos y del modo de pantalla establecido mediante la última instrucción SCREEN. Si su sistema tiene un adaptador EGA, VGA o MCGA, utilice la instrucción PALLETTE para cambiar las asignaciones de color correspondientes a los atributos de color.

PRINT Y LPRINT

PRINT escribe los datos a la pantalla o a un archivo.

LPRINT imprime datos en la impresora LPT1.

SINTAXIS DEL COMANDO

PRINT [#filenumber%,] [expressionlist] [{; |,}]

LPRINT [expressionlist] [{; |,}]

COMENTARIO

Filenumber% Es el número de un archivo abierto. Si usted no especifica el número de archivo, IMPRESIÓN escribe a la pantalla.

Expressionlist A unos o más expresiones numéricas o de la secuencia imprimir.

{;|, } Se determina dónde la salida siguiente comienza:

; Los medios imprimen inmediatamente después del valor pasado.

, Los medios imprimen en el comienzo de la zona siguiente de la impresión. Las zonas de la impresión son 14 caracteres de par en par.

LOCATE

LOCATE mueve el cursor en la pantalla a la posición especificada.

CSRLIN devuelve la posición actual de la fila donde se encuentra el cursor.

POS devuelve la posición actual de la columna donde se encuentre el cursor.

SINTAXIS DEL COMANDO

LOCATE [fila%] [,[columna%] [,[cursor%] [,inicio% [,fin%]]]]

CSRLIN

POS(expresión)

COMENTARIO

- Fila% y columna%; El número de la fila y columna a la que se moverá el cursor.
- Cursor%; Especifica si el cursor está visible:

  0 = invisible, 1 = visible
- Inicio% y fin%; Expresiones de enteros entre 0 y 31 que especifican la primera y última línea de exploración del cursor. Podrá cambiar el tamaño del cursor modificando las líneas de exploración.
- Expresión; Cualquier expresión.

UNTIL

Repite un bloque de instrucciones mientras una condición tenga el estado verdadero, o hasta que una condición adquiera el estado verdadero.

SINTAXIS DEL COMANDO

DO [{WHILE | UNTIL} condición]

[bloqueinstrucciones]

LOOP

DO

[bloqueinstrucciones]

LOOP [{WHILE | UNTIL} condición]

COMENTARIO

- Condición.- Una expresión numérica que Basic evalúa como verdadero (no cero) o falso (cero).

SHARED

SHARED da el acceso de los procedimientos y las variables del modulo=nivel. STATIC hace una variable local a una función o a un procedimiento y preserva su valuación entre las llamadas.

SINTAXIS DEL COMANDO

SHARED variable[()] [AS type] [,variable[()] [AS type]]

STATIC variable[()] [AS type] [,variable[()] [AS type]]

COMENTARIO

Variable el nombre de la variable del modulo-nivel de la parte o de la variable hace parásitos atmosféricos. Los nombres de variables pueden consistir hasta 40 y los caracteres deben comenzar con una letra. Los caracteres válidos son A-z, 0-9, y el período (.).

AS type declara el tipo de datos de la variable (NÚMERO ENTERO, LARGO, SOLO, DOBLE, SECUENCIA, o un tipo definido por el usuario).