Indice

1 - Introducción

El intérprete GFA-BASIC 3.0 le brinda un amplio lenguaje de programación con un confortable entorno de desarrollo. Este dialecto avanzado de Basic dispone de un editor eficiente y veloz, que sostiene la creación de programas estructurados.

El intérprete ofrece una confortable posibilidad de tratamiento de error a través de comandos especiales para la búsqueda y eliminación de errores (por ej. la procedure TRON y TRACE$). Los lenguajes modernos de programación se caracterizan por los diferentes comandos que permiten escribir programas estructurados. El editor sostiene además la programación estructurada mediante sangrías automáticas de comandos en estructuras de repetición y en condiciones. Además, los subprogramas se pueden representar en el listing del programa por medio de una anotación para reservar el lugar, en la cual se puede "desplegar" o "plegar" el subprograma si se pulsa una tecla.

En cuanto a las instrucciones condicionadas se agregan a los comandos IF-ELSE-ENDIF, que ya existían en versiones anteriores de GFA-BASIC, otros comandos para bifurcaciones múltiples (ELSE IF, SELECT-CASE).

Para poder formular subprogramas existe la posibilidad de unificar procedimientos y funciones (nuevo en el 3.0). En este caso se pueden transferir tanto valores de variables como las mismas variables. Además de los tipos de estructuras de repetición conocidos de versiones anteriores de GFA-BASIC (FOR-NEXT, REPEAT-UNTIL, WHILE-WEND y DO-LOOP) se dispone ahora de comandos ampliados para loops como DO-UNTIL, DO-WHILE, LOOP-UNTIL y LOOP-WHILE.

La posibilidad de llamar rutinas del sistema operativo (GEMDOS,BIOS,XBIOS) otorga una programación cercana al sistema. Muchas de estas funciones también están a disposición en forma de instrucciones simples. Además es posible la programación manejada con interrupt a través de EVERY y AFTER. Subprogramas escritos en assembler y C se pueden integrar con comandos como por ej. RCALL, C: y MONITOR.

Las principales rutinas de VDI y todas las funciones AES (por ej. funciones de administración de menús, de ventanas y de formularios) se pueden aplicar como funciones incorporadas. Junto con comandos que simplifican la incorporación de GEM, todos estos comandos facilitan el desarrollo de programas que deben tener una superficie de trabajo GEM.

El intérprete GFA-BASIC 3.0 dispone de una verdadera aritmética de números enteros que brinda una alta velocidad de cálculo, así como de una aritmética de punto flotante con una elevada precisión de cálculo (13 decimales).

Comparado con las versiones 2.xx del GFA-BASIC se agregaron otros tipos de variables (BYTE, WORD) y operaciones de bits (BCLR, BSET, BTST, BCHG, SHL, SHR, ROL, ROR etc.). Programas de gráficos pueden hacer uso de rutinas de Line-A, las cuales están implementadas como comandos.

El concepto de este manual

Este manual comienza con una breve descripción del lenguaje de programación GFA-BASIC 3.0. Luego sigue una explicación de la estructura de este manual (la cual está leyendo justamente) y una introducción al uso del intérprete GFA-BASIC. El capítulo finaliza con una descripción de los puntos que se deben respetar cuando se asumen programas escritos en versiones anteriores del GFA-BASIC. El siguiente capítulo desbribe los comandos para el uso del editor. Todos los demás capítulos contienen esencialmente una descripción de los comandos, instrucciones y funciones del GFA-BASIC 3.0. Estós se han ordenado de acuerdo a su contenido, explicándose conjuntamente conceptos emparentados (por ej. MIN y MAX). En el anexo se encuentra un resumen alfabético de los comandos y funciones, con las referencias a las correpondientes páginas del manual. Estas tienen la siguiente estructura:

- Indicación de la sintaxìs
- Descripción de los tipos permitidos de parámetros
- Texto aclaratorio
- Ejemplo

En la indicación de la sintaxis se han marcado entre corchetes los parámetros opcionales.

```
LEFT$(a$[,x])
```

En el GFA-BASIC hay comandos o instrucciones y funciones. Los comandos o instrucciones no retornan ningún valor:

```
LINE 100,100,200,200
```

Las funciones en cambio devuelven un valor. Este valor puede ser asignado a alguna variable, incorporado al cálculo de una expresión o evidenciado por medio de PRINT. A continuación algunos ejemplos:

```
a = ASC("A")
b = ASC("A") + 32
PRINT ASC("65")
PRINT ASC("A")
```

En este manual no se aclara en la indicación de sintaxis que las funciones devuelven un valor. Esto surge de la descripción de las funciones. Como indicación de sintaxis solo se indica por ej. ASC(a$).

Parámetros de libre elección, de los cuales pueden figurar cualquier cantidad en una instrucción (por ej. DATA), también se presentan entre corchetes. En éstos figuran dos parámetros, seguidos de tres puntos, por ej.:

```
DATA[x,y,...]
```

La indicación del tipo de parámetro permitido figura bajo la indicación de la sintaxis. Para estos tipos se emplean las siguientes abreviaturas:

avar Variable aritmética. Se trata de una variable numérica de cualquier tipo.

aexp Expresión aritmética (arithmetic expression). Esta es una expresión de cualquier grado de complejidad, que produce un número. También puede tratarse de una constante (un número) o de una variable (las variables son un grupo parcial de las expresiones). Ejemplos de expresiones aritméticas son:

```
a%
3
2+a%+ASC("A")
```

svar Variable string. Esta es una variable de cadena de caracteres. Este tipo de variables tiene la terminación $.

sexp Expresión de cadenas de caracteres (string expression). Esta expresión puede tener cualquier grado de complejidad y debe dar un string. Puede tratarse en este caso de una constante (un texto entre comillas) o una variable de cadena de caracteres. Ejemplos de expresiones string son:

```
a$
"Test"
a$+"Prueba"+"LEFT$("Manual",4)
```

ivar Variable entera (integer variable).

iexp Expresión entera (integer expression).

bexp Expresión lógica (boolean expression).

Es importante, que en algunos parámetros numéricos no se indique cualquier tipo de variable numérica. El principal ejemplo para esto son las direcciones. Las direcciones se deben indicar por lo menos en una variable de cuatro byte; o sea no se permiten variables boolean, byte o word. Después de la descripción de los parámetros permitidos sigue la explicación de los comandos e instrucciones. Allí se explica el significado de éstos y sus diferentes parámetros .

La explicación de un comando o instrucción se finaliza con uno o varios ejemplos. Cada uno de estos ejemplos se debe ingresar en el editor e iniciar a través de RUN (Shift + F10 o activación de RUN en la zona de menú). Detrás de cada comando e instrucción se indica qué efecto produce.

Este concepto de descripción de comandos e instrucciones solo se modifica algo en el capítulo sobre bibliotecas AES. Aquí se indica el nombre del comando o instrucción correspondiente, luego sigue la descripción de su función. Recién entonces se indica la sintaxis de la instrucción con la explicación de los diferentes parámetros. Recién al final del capítulo sobre bibliotecas AES se presentan varios programas más extensos en forma de ejemplos.

Esta modificación se debe a que muchos comandos e instrucciones de ese capítulo solo trabajan convenientemente con otros comandos e instrucciones. El uso de las diferentes instrucciones es demostrado en los respectivos programas de ejemplo.

El manual finaliza con una colección de tablas y una lista ordenada alfabéticamente de todos los comandos e instrucciones, en la cual se indica el número de página donde se describe el respectivo comando o instrucción.

El primer conocimiento con el GFA-BASIC 3.0

Este apartado se ha escrito para aquellos usuarios del GFA-BASIC 3.0 que hasta el momento no han tenido ninguna experiencia con lenguajes de programación. Aquellos que ya han trabajado con versiones anteriores, pueden saltear este apartado.

El disco del programa de GFA-BASIC no tiene protección de copia. Usted debe crear en primer lugar una copia de seguridad del disco original. La descripción del proceso de copiado la encuentra en el manual de su ordenador. Coloque entonces la copia del disco del programa en la disquetera e arranque el intérprete GFA-BASIC.

En la pantalla aparece entonces el editor, en el cual puede escribir sus programas. ESCRIBA ahora las siguientes líneas del programa y pulse la tecla Return después de ingresar cada línea.

Para ello no necesita escribir los espacios en blanco, que figuran delante de algunas instrucciones. El editor se encarga de ubicar automáticamente las instrucciones dentro de las estructuras de repetición. Tampoco es necesario respetar las minúsculas y mayúsculas al ingresar líneas de sentencia. Si abandona una línea con Return, automáticamente se escriben las minúsculas y mayúsculas.

```
DEFFILL 1,0
REPEAT
   WHILE MOUSEK=1
     PBOX MOUSEX,MOUSEY,MOUSEX+30,MOUSEY+30
   WEND
UNTIL MOUSEK=2
```

En el extremo derecho superior de la pantalla encuentra la palabra "Run". Para iniciar el programa señale con la flecha del ratón esta palabra y pulse la tecla izquierda del ratón.

Entonces aparece una pantalla blanca, sobre la cual se visualiza la flecha del ratón. Si ahora pulsa la tecla izquierda del ratón y desplaza el ratón sobre la mesa, puede dibujar sobre la pantalla. Como "pincel" se usa un rectángulo. La pulsación de la tecla derecha del ratón produce la finalización del programa. Entonces aparece un ventana de alerta en la pantalla, que contiene el mensaje "Fin del programa". Ubique el cursor en Return y pulse la tecla izquierda del ratón. A continuación se vuelve a encontrar en el editor.

Cómo funciona este programa? La instrucción PBOX permite dibujar rectángulos en el medio del programa. La primer instrucción del programa(DEFFILL 1,0) indica que se rellene el interior de cada rectángulo. Los cuatro parámetros definen los extremos de este rectángulo.

Las variables MOUSEK, MOUSEX y MOUSEY contienen informaciones sobre el ratón. MOUSEK dice, qué tecla se está pulsando en cada momento. En este caso significa MOUSEK=1 la tecla izquierda y MOUSEK=2 la derecha. En MOUSEX y MOUSEY figura la posición x e y del del cursor del ratón sobre la pantalla (comparar Representación gráfica, DEFMOUSE).

Los restantes comandos e instrucciones (REPEAT, WHILE, WEND, UNTIL) son instrucciones de loops. El loop de las instrucciones WHILE

MOUSEK=1 y WEND se puede traducir como la formulación "repite, hasta que se pulse la tecla izquierda del ratón". El loop exterior de las instrucciones REPEAT y UNTIL MOUSEK=2 significa "repite, hasta que se pulse la tecla derecha del ratón". Como no figura ninguna otra instrucción detrás del comando UNTIL, el programa finaliza después de abandonar el loop REPEAT-UNTIL.

Ingrese la siguiente línea de sentencia errónea detrás de la última línea del programa, en la cual falta la i de la instrucción PRINT:

```
prnt "Test"
```

Si pulsa la tecla Retrun para confirmar la instrucción y abandona la línea, suena una señal acústica y en la segunda línea de la pantalla aparece el mensaje "Error de sintaxis".

El editor revisa por lo tanto durante la creación del programa si las instrucciones se ingresan correctamente. Vaya ahora con el cursor a la letra r y pulse tres veces la tecla Delete. En la línea de programa figura ahora únicamente:

```
p"Test"
```

Si ahora pulsa la tecla Return, puede abandonar la línea, pues la letra p se reconoce automáticamente como abreviatura para la palabra de la instrucción PRINT.

Esto debería ser suficiente como primer contacto con el intérprete.

El editor

Fundamentos

El editor del GFA-BASIC 3.0 no es un editor de texto convencional, sino que fue creado especialmente para el desarrollo de programas. Por ejemplo, durante la creación del programa ya se reconocen sentencias sintácticamente erróneas. Además, se ubican automáticamente los comandos dentro de estructuras de repetición (loops) o con instrucciones condicionadas y las abreviaturas de comandos se transfieren al nombre completo del comando (por ej. p en PRINT).

Al escribir un listado de programa siempre se realiza un control de sintaxis, cuando se debe abandonar una línea. Si la sentencia en esta línea no es correcta desde el punto de vista sintáctico , entonces aparece en la segunda línea el mensaje "Syntax Error". Fijando el símbolo de comentario (') al comienzo de la línea, ésta se transforma en comentario. Después se puede abandonar la línea.

En cada línea del programa sólo se puede ingresar una sentencia. Detrás de esta sentencia puede aparecer además un comentario. Esta sentencia no puede tener más de 255 caracteres. Cuando una línea supera los 80 caracteres, su comienzo desaparece en el flanco izquierdo de la pantalla. Por lo tanto, sólo se hace el "scrolling" de esta única línea.

Al abandonar una línea de sentencia, se realiza un control de sintaxis (ver más arriba) y se da formato a la línea. O sea, por ej. hay que eliminar espacios en blanco superfluos (por ej. de 2 + 2 resulta entonces 2+2) y escribir las palabras de las sentencias y los nombres de las variables según el DEFLIST conectado. Si se preseleccionó el DEFLIST 0, por ej., todas las palabras de las sentencias se escriben en mayúscula y todos los nombres de las variables en minúscula.

El bloque de teclas del cursor

El cursor se puede dirigir a través del bloque de teclas del cursor. La instalación de las teclas es:

Flecha hacia la izquierda:		el cursor se desplaza un carácter a la izquierda
Flecha hacia la derecha	:	el cursor se desplaza un carácter a la derecha
Flecha hacia arriba	:	el cursor se desplaza una línea hacia arriba
Flecha hacia abajo	:	el cursor se desplaza una línea hacia abajo

Los movimientos del cursor están sometidos a algunas limitaciones. Como máximo se puede ubicar un carácter detrás del último carácter de una línea y una línea por debajo de la última línea del programa.
Cuando el cursor pasa a una línea, que es más corta que la posición actual de columna del cursor, el cursor salta al final de esta línea, en lo demás mantiene su posición actual de columna (a diferencia de los editores de versiones más antiguas de GFA-BASIC).

También es posible ubicar el cursor con el ratón. Para ello se señala con la flecha del ratón la posición deseada y se oprime la tecla

izquierda del ratón.

Al pulsar la tecla de Insert se introduce una línea en blanco entre la línea sobre el cursor y la línea del cursor, si no se ha realizado ningún cambio en la línea actual. El cursor se ubica al comienzo de esta línea en blanco. Con Clr-Home se lleva el cursor al márgen superior izquierdo por debajo de la zona del menú , con Control+Clr/Home se lleva el cursor al comienzo del listado del programa.

Con ayuda de la tecla Undo se pueden revertir cambios en una línea del programa, en la medida que no se ha abandonado todavía la línea. La tecla Help posibilita "desdoblar y doblar" procedimientos en el listado del programa.

Quiere decir: cuando se ubica el cursor en una línea, en la cual aparece el comando PROCEDURE y luego se pulsa la tecla Help, entonces desaparecen todas las líneas de sentencia en el listado hasta el siguiente RETURN (inclusive). En lugar de eso se coloca una flecha y un espacio en blanco '> ' delante del comando PROCEDURE.

En esta situación tampoco se pueden realizar cambios en los nombres del procedimiento o la lista de parámetros del subprograma. Para poder volver a ver las líneas de sentencia entre PROCEDURE y RETURN, se desplaza el cursor a la línea , que comienza con la flecha y se pulsa nuevamente la tecla Help o se suprime el carácter '>'.

Este " Doblar" (Folding) de un subprograma hace posible crear listados breves y claros, en los cuales solo se "desdobla" el subprograma, con el cual se trabaja en ese momento. Un programa con procedimientos doblados puede ser así, por ej.:

```
init
main_menu
'
> PROCEDURE INIT
> PROCEDURE menu_principal
> PROCEDURE zona de menú
> PROCEDURE cargar
> PROCEDURE almacenar
> PROCEDURE elaborar
> PROCEDURE buscar información
> PROCEDURE mostrar información
```

Con un procedimiento desdoblado el programa es entonces el siguiente:

```
init
main_menu
'
> PROCEDURE init
> PROCEDURE menu_principal
> PROCEDURE zona de menú
PROCEDURE cargar
  FILESELECT "*.RSC","",rsc_fichero$
  IF NOT EXIST(rsc_fichero$)
     ALERT 1,"No existe fichero !",1,"Interrupción",r%
  ELSE
     IF RSRC-LOAD(rsc_fichero$)=0
       ALERT 1, "Error en la carga del fichero !",1, "Interrupción",r%
```

```
      END
    ENDIF
  ENDIF
RETURN
> PROCEDURE almacenar
> PROCEDURE elaborar
> PROCEDURE buscar información
> PROCEDURE mostrar información
```

El teclado numérico

El teclado numérico sirve normalmente para ingresar números y algunos otros caracteres. Pero también se puede usar en combinación con la tecla de control. Su instalación corresponde entonces a la instalación en teclados con NUMLOCK conectado (por ej. en PC's). La instalación es:

Control y 4 Desplaza cursor un carácter a la izquierda
Control y 6 Desplaza cursor un carácter a la derecha
Control y 8 Desplaza cursor una línea hacia arriba
Control y 2 Desplaza cursor una línea hacia abajo
Control y 7 Salto al comienzo del programa
Control y 1 Salto al final del programa
Control y 9 Paso a la página anterior
Control y 3 Paso a la página siguiente
Control y 0 Corresponde a Insert
Control y . Corresponde a Delete.

El teclado numérico también puede trabajar en un modo, en el cual éstas teclas son efectivas sin Control. La conmutación se hace a través de otras (combinaciones) de las teclas de control del teclado numérico.

Control y - , así como Control y (corresponden a NUMLOCK. Entonces al pulsar una tecla del teclado numérico resulta el comando indicado anteriormente con Control. El modo NUMLOCK se puede volver a anular si se vuelve a pulsar la misma tecla.

Otras teclas editoras en el teclado principal

Con la tecla DELETE se puede suprimir el carácter que está en la posición del cursor; el resto de la línea se acopla sin dejar espacios en blanco. La tecla Backspace suprime el carácter, que se encuentra a la izquierda del cursor y acopla el resto de la línea sin dejar espacios en blanco.

Al pulsar la tecla tab (tabulador) el cursor salta a la siguiente posición de tabulación a la derecha. Estas posiciones se encuentran en intervalos de ocho caracteres. Control + tab lleva el cursor a la siguinete posición de tabulación a la izquierda.

Las teclas Return y Enter hacen saltar el cursor al comienzo de la siguiente línea. Pulsando la tecla Escape, se pasa al modo directo.

Comandos de control

En los párrafos anteriores ya se han mencionado muchos comandos de Control. Aquí se han de resumir una vez más estos comandos y los comandos de las teclas de Control no mencionadas allí (excepto los comandos de Control, que trabajan junto con el teclado numérico):

Control + Delete suprime la línea en la que está el cursor
Control + U (undelete)

inserta la línea que se suprimió como última con Control + Delet o Control + Y. Esto sirve, por un lado, para restaurar líneas, que se han suprimido sin querer. Por otro lado, así se pueden desplazar o copiar fácilmente líneas individuales (por ej. para copiar: sostener la tecla de Control y pulsar Delete y dos veces U).

Control+Y	Suprime la línea que está en la posición del cursor
Control+N	Inserta una línea en blanco por encima de la línea de órden que está en la posición del cursor (como Insert), también cuando se han llevado a cabo modificaciones en la línea.
Control+Q	Llama al menú de bloque (igual que la tecla de función F4).
Control+B	Marca el comienzo del bloque (bloque).
Control+K	Marca el fin del bloque.
Control+R	Pasa a la página anterior.
Control+C	Pasa a la página siguiente.
Control+E	Sustituye texto
Shift+Control+E	Sustituye preguntando con el string de buscar y sustituir
Control+F	(find), busca texto.
Shift+Control+F	Busca texto preguntando con el string de buscar.
Control+cursor izquierda	Salta al comienzo de la línea.
Control+cursor derecha	Salta al final de la línea.
Control+cursor arriba	Pasa a la página anterior.
Control+cursor abajo	Pasa a la página siguiente.
Control+Clr+Home	Salta al principio del programa
Control+Z	Salta al final del listado del programa.
Control+Tab	Salta a la posición de tabulación a la izquierda.
Control+G (goto)	Abre el indicador de número de línea para ingresar un número de línea (se pasa entonces a la correspondiente línea).

Un grupo especial de Comandos de control hace posible fijar y acceder a marcas en el editor. Estas marcas valen solo para el editor, o sea no tienen nada que ver con las marcas que utiliza GOTO o RESTORE. Estas marcas del editor se pueden ubicar en la posición del cursor pulsando Control y un Número del teclado principal. A continuación se puede saltar a la marca deseada oprimiendo Alternate y este número.

Las combinaciones de teclas de Alternate con los números 7-9 y 0 están preconfiguradas. Al pulsar Alternate y 7 se salta a la última posición del cursor antes de pasar al modo directo o antes del último comienzo de programa. Alternate y 8 salta a la posición, en la que se encontraba el cursor al comienzo del editor. Con Alternate y 0 se salta a la última posición del cursor, en que se realizó una modificación.. Alternate y 9 salta a la posición, en la que se inició la última operación de búsqueda.

La zona de menú y las teclas de función

En las dos líneas superiores de la pantalla del editor se encuentran dos líneas del menú. Bien a la izquierda aparece el símbolo ATARI, que cuando se activa desdobla un menú Pull-Down. Este contiene los títulos del menú del símbolo ATARI y el GFA-BASIC.

Cuando se activa el item del menú GFA-BASIC3.00 se visualizan los números de versión con las dos posibilidades de elección: editor y menú. Si se activa el botón del editor, se vuelve al editor GFA-BASIC. Al elegir el menú se vuelve al menú Pull-Down.

El título del menú GFA-BASIC contiene los siguientes ítems de menú:

Save — Salta al editor, luego se visualiza un File-Select-Box, y se puede ingresar un nombre de fichero, bajo el cual se ha de guardar el programa actual.

Load — Salta al editor, después de lo cual se puede ingresar el nombre de un fichero en el File-Select-Box que se visualiza.

Deflist — Posibilita la conexión del modo Deflist a través de un Alert-Box (comparar apartado Casos especiales).

Nombres nuevos — Con ayuda de este ítem del menú se puede conectar un modo, en el cual se debe confirmar el ingreso de nuevas variables en el programa actual. Este modo está desconectado en la preselección. Cuando se emplea una variable que no se ha ingresado aún o también un nombre de un procedimiento o de una función y este modo está activo, aparece un Alertbox, en el cual se puede indicar si se quiere incorporar esta nueva variable al programa o si se trata de un error de escritura (en este caso se visualiza el mensaje de error "Syntax Error").

En el flanco derecho de la zona de menú se encuentra en la parte superior un reloj y debajo de este la indicación del número de línea de texto actual. El manejo de estos dos elementos de la zona de menú se explicará más adelante.

Entre el símbolo ATARI y el reloj se visualizan veinte palabras de comando, de a dos una sobre la otra, las cuales se pueden activar con el ratón o a través de las teclas de función. La línea inferior de las palabras de comando se puede activar pulsando una tecla de función; los comandos de la línea superior pulsando la tecla Shift y la correspondiente tecla de función. A la palabra de comando en la parte inferior izquierda pertenece por ej. la tecla F1, la palabra de comando por encima de ésta (Save) se llama con Shift+F1.

Debajo del símbolo ATARI hay lugar para otros dos caracteres. Allí se visualiza el estatus de la tecla CAPSLOCK y NUMLOCK. Si se pulsa la tecla CAPSLOCK se visualiza a la izquierda debajo del símbolo ATARI una flecha que apunta hacia arriba. Cuando se pulsa la tecla NUMLOCK, o sea Control+(, se visualiza a la derecha debajo del símbolo ATARI el símbolo de elevación a potencia ^. El modo NUMLOCK y CAPS-LOCK también se pueden seleccionar activando la tecla del ratón izquierda debajo del símbolo ATARI.

Los items del menú tienen el siguiente significado:

Load F1

El comando Load hace posible cargar un programa GFA-BASIC3.0. El formato empleado aquí emplea los llamados tokens de sentencia. Un fichero de programa de este tipo se puede cargar y almacenar con gran velocidad . Para este formato se usa normalmente el sufijo de fichero .GFA.

Las versiones más antiguas de GFA-BASIC usan otro formato token. Por eso,sus programas en la versión antigua se deben almacenar con Save,A y cargar en GFA-BASIC con Merge. Luego se pueden almacenar en 3.0 con Save y cargar con Load.

Save Shift+F1

Se visualiza un File-Select-Box, en el cual se puede indicar un nombre. Bajo este nombre se puede almacenar entonces el programa, que se encuentra en ese momento en el editor. Aquí se usa el formato tokenizado bajo Load. El sufijo del fichero conectado previamente se llama .GFA. Se agrega automáticamente al nombre del fichero, si el usuario no indica sufijo.

Si ya existe un fichero con el nombre ingresado, éste se almacena con el mismo nombre de fichero con el sufijo .BAK.

Merge F2

Con este comando se puede insertar en el programa actual un fichero en el formato ASCII. La inserción se realiza antes de la línea en que está el cursor. El sufijo del fichero conectado previamente es .LST. Los programas de versiones más antiguas de GFA-BASIC se almacenen de acuerdo con la versión antigua con Save,A y se vuelven a cargar en GFA-BASIC con Merge.

= = > Las líneas del programa con este sufijo surgen con el Merge = = > de un programa, que el Intérprete no entiende.

Save,A Shift+F2

Con Save,A se puede almacenar el programa actual en formato ASCII. Un fichero almacenado en este formato se puede volver a cargar con Merge. El sufijo del fichero conectado previamente es .LST. Se agrega automáticamente, si el usuario no ingresa ningún sufijo.
Si ya existe un fichero bajo el nombre ingresado, éste se almacena bajo el mismo nombre con el sufijo .BAK.
Llist F3

Este comando inicia la impresión del programa que se encuentra en el editor. El formateo de la impresión se puede influenciar con las llamadas instrucciones de punto. Estas instrucciones se insertan en el listado del programa igual que líneas de sentencia normales. Son (x es una anotación que se usa para indicar cuántos números aparecerán):

.ll xx - longitud máxima de la línea
.pl xx - longitud máxima de la página
- pasar a la siguiente página

	- conditional page
	- enumeración de las páginas
	- márgen izquierdo
.ff xxx	- cadena de caracteres form-feed (para impresoras que tienen otros valores de form-feed, se puede ingresar una cadena de sustitución. Previamente se conectó .ff\012)
.he cabecera	- texto de la línea de cabecera
.fo pié	- texto de la línea de pie de página
.lr xx	- márgen izquierdo
.l-	- Con esta indicación se logra que las líneas siguientes no aparezcan en el listado. El listado se puede volver a conectar mediante un .l+ que aparece más adelante en el programa.
.l+	- ver .l-
.n1 a .n9	- conecta la enumeración de la líneas con 1 a 9 lugares.
.n0	- desconecta la enumeración de las líneas.

En texto de la línea de cabecera y al pie de página así como en el string FF se pueden insertar otras anotaciones que indican cuántos números aparecerán.

\xxx	- Carácter con el ASCII-Code xxx
\d	- Fecha
\t	- hora
#	- número de página

Para poder evidenciar los caracteres especiales # y \, hay que fijar otro carácter \ delante. Por lo tanto, la combinación de caracteres \\ evidencian un \ en la impresora y la combinación \# un #.

Quit Shift+F3

Hace posible abandonar intérpretes GFA-BASIC.

Block F4

Si no se marcó ningún bloque y se selecciona éste ítem del menú, entonces se visualiza en la zona de menú el texto "Block???", para indicar, que la elección del comando bloque no tiene sentido en esta situación. Pero si se marcó un bloque, se visualiza un menú en la línea superior de la pantalla. Los ítems individuales de este menú se pueden activar con el ratón o pulsando una letra de comando. Los ítems del menú son (Entre paréntesis aparece siempre la tecla, con la que se puede llamar el ítem del menú):

Copy Copia el bloque en la posición actual del cursor. En este caso queda marcado el bloque.

Move Desplaza el bloque a la posición actual del cursor. En este caso se elimina la marca del bloque.

Write Almacena el bloque como fichero ASCII.

Llist Imprime el bloque.

Start Salta al comienzo del bloque.

End Salta al final del bloque.

^Del Suprime el bloque (Control-D).

Hide: Elimina la marca del bloque.

Pulsando otra tecla (o la tecla del ratón) fuera de esta zona se elimina el menú del bloque.

New Shift+F4

Suprime el programa que se encuentra en el editor.

BlkEnd F5

Con BlkEnd se puede marcar la línea en que está el cursor como final de bloque. Si la marca del comienzo del bloque se encuentra sobre esta línea, entonces el bloque se enmascara con una matriz de puntos en la versión monocromática. En la versión color se representa el bloque marcado en otro color. Este comando también se puede ejecutar pulsando Control+K.

BlkSta Shift+F5

Con BlkSta se marca la línea en que está el cursor como comienzo del bloque. Si detrás de esta línea se fijó anteriormente un final de bloque, la zona del bloque se enmascara con una matriz de puntos. Este comando también se puede ejecutar pulsando Control+B.

Find F6

Después de inicializar el comando Find se puede ingresar un texto a buscar . La búsqueda comienza en la línea en que está el cursor. La búsqueda se puede volver a llamar pulsando Control+F o Control+L, sin que se visualice la pregunta del texto a buscar.

Cuando se encontró el texto buscado, el cursor aparece al comienzo de la línea que contiene el texto buscado; cuando no se encontró aparece el cursor al final del programa.

Cuando se vuelve a llamar el comando Find, entonces aparece en la línea, en la que se puede ingresar el string de buscar, el último string de buscar usado. Al ingresar el texto a buscar se dispone de las siguientes posibilidades de edición:

Cursor izquierda	El cursor se desplaza un carácter a la izquierda (en la medida en que no se encuentra bien a la izquierda).
- derecha	El cursor se desplaza un carácter a la derecha (en la medida en que no se encuentra al final de la cadena de búsqueda).
Delete	Suprime el carácter de la cadena de búsqueda, que está debajo del cursor, el resto de la cadena de búsqueda se acopla de derecha a izquierda.

Backspace	Suprime el carácter a la izquierda del cursor (en la medida en que no se encuentra ya en el márgen izquierdo) y envuelve el resto de la cadena de búsqueda.
Escape	Suprime el campo de entrada
Return/Enter	Confirma el texto de búsqueda ingresado y comienza a buscar.

El comando Find también se puede llamar con la combinación de teclas Shift+Control+F o Shift+Control +L. En procedimientos "doblados" no se encuentra texto (ver tecla Help en el párrafo teclado del cursor).

Replace Shift+F6

Este comando sirve para sustituir un texto por otro. Después de llamarlo, se pregunta por el texto que se quiere sustituir. Luego se pregunta, cual es el texto que lo sustituirá. Entonces comienza en el programa la búsqueda del texto que se quiere sustituir desde la posición del cursor. Cuando se encuentra el texto buscado, el cursor salta al comienzo de la línea correspondiente.

Entonces se puede realizar la sustitución pulsando la combinación de teclas Control+E. Si se sigue pulsando la combinación de teclas se busca la siguiente parte del texto a sustituir o, si se encuentra en la posición actual del cursor, la sustituye.

El comando Replace también se puede llamar con la combinación de teclas Shift+Control+E. Las posibilidades de edición, de las que se dispone al ingresar el texto de búsqueda o de sustitución, se han señalado en la descripción del comando Find. En procedimientos "doblados" no se encuentra ningún texto (ver tecla Help en el párrafo teclado del cursor).

Pg Down F7

"Da vuelta la página" del texto del programa "enrollando" el texto de la pantalla hacia abajo. También se puede llamar con Control+C.

Pg Up Shift+F7

"Da vuelta la página" del texto del programa, "enrollando" el texto de la pantalla hacia arriba. También se puede llamar con Control+R.

Insert/Overwr F8

Pasa del modo INSERT (inserción de texto) a modo OVERWRITE (cada nuevo carácter es tecleado sobre el carácter que está en la posición del cursor, los caracteres sobre los que se teclea son suprimidos) y viceversa.

Txt16/Text8 Shift+F8

Este comando solo está a disposición en la definicion monocromatica. En el modo 16 se representan letras con un alto de 16 pixel, o sea que en la pantalla se visualizan simultáneamente 23 líneas. En el modo 8 el alto de las letras es de solo 8 pixel, pero en la pantalla se visualizan simultáneamente 48 líneas. Con Txt16/Text8 se puede conmutar

entre estos dos modos.

Flip F9

Con este comando se puede conmutar a la pantalla de salida, donde se visualiza la situación que dejó el último programa llamado. Pulsando una tecla (también la tecla del ratón) se puede volver a la pantalla del editor.

Direct Shift+F9

Con este comando accede al modo directo. De esta manera puede ingresar comandos de GFA-BASIC para que se ejecuten inmediatamente después de pulsar la tecla Return o Enter. Algunos comandos, por ej. comandos de estructuras de repetición (loops), no se pueden llamar en el modo directo. A este modo también se puede acceder pulsando la tecla Escape.

En el modo directo además es posible volver a llamar las últimas ocho sentencias (que se ingresaron en el modo directo) subiendo y bajando el cursor con las teclas.

Pulsando la tecla Undo se visualiza la última sentencia. Con las combinaciones de teclas Control+Cursor izquierda o bien Control+Cursor derecha se pueden alcanzar el comienzo y final de la línea. Además, pulsando la tecla Insert se puede conmutar entre el modo de inserción y normal (por defecto).

El modo directo se puede volver a abandonar pulsando la tecla Escape seguida de Return. En forma alternativa también se puede oprimir simultáneamente Control+Shift+Alternate.

Para llamar instrucciones de varias líneas, se puede escribir en el editor un procedimiento, que contiene estas sentencias. En el modo directo se puede llamar entonces a este procedimiento.

Test F10

Después de la ejecución de esta instrucción se examina si todas las estructuras de repetición (loops), los subprogramas y las instrucciones condicionadas del programa actual cierran. O sea, se lleva a cabo una prueba de estructura.

Con Control, la tecla Shift izquierda y Alternate se puede interrumpir un programa en ejecución.

Run Shift+F10

Inicia la ejecución del programa que se halla en el editor. Si este programa contiene un error estructural, por ej. un loop FOR-NEXT no cerrado, se visualiza el correspondiente mensaje de error y no se inicia la ejecución del programa.

Control+Shift+Alternate

Con la combinación de teclas Control+tecla Shift izquierda+Alternate se puede interrumpir el programa en ejecución.

El indicador de línea y de la hora

En la parte inferior derecha de la zona de menú se encuentra el

indicador de la línea en que se encuentra el cursor. En este indicador de línea también se puede incorporar un número de línea, de tal manera que el cursor salta a la línea correspondiente. Con este fin se puede activar la zona de indicación mediante el ratón o se puede inicializar con el comando Control+G.

En este campo solo se pueden ingresar números. Las posibilidades de edición para el ingreso de los números de línea deseados son:

Cursor izquierda	El cursor se desplaza un carácter a la izquierda (en la medida en que no se encuentra bien a la izquierda).
- derecha	El cursor se desplaza un carácter a la derecha (en la medida en que no se encuentra bien a la derecha).
Backspace	Tiene la misma función que Cursor izquierda.
Escape	Suprime el campo de entrada.
Return/Enter	Confirma el número ingresado.

Sobre el indicador del número de línea se encuentra un reloj, que indica la hora del sistema. Después de activar el reloj con la flecha del ratón, existe la posibilidad de volver a ajustar el reloj. Las posibilidades de edición para el ingreso de la hora son las mismas que para el campo indicador del número de línea. Solo la tecla Escape tiene aquí otro significado, ya que interrumpe el ingreso manteniendo para la hora el valor viejo.

Casos especiales

```
DEFBITf$
DEFBYTf$
DEFINTf$
DEFWRDf$
DEFFLTf$
DEFSTRf$
```

f$: constante string o alfanumérica

La instrucción DEFxxx sirve para declaración simplificada de variables. En este caso xxx es una anotación que indica el número de abreviaturas de tipos de variables, que se enumeran y explican en la siguiente tabla:

DEFxxx	f$	Sufijo	Todas las variables...
DEFBIT	"B"	!	con la letra inicial 'b' se declaran variables booleanas.
DEFBYT	"by"	I	con las letras iniciales 'by' se declaran como entera de un byte.
DEFWRD	"w"	&	con la letra inicial 'w' se declaran como entera de dos byte con signo.

DEFINIT	"i-k,m-p"	%	con las letras iniciales i' a 'k'y 'm' a 'p' se declaran como enteras de 4 byte con signo.
DEFFLT	"x-z"	#	Con las letras iniciales 'x' a 'z'se declaran como valores de punto flotante de 8 byte.
DEFSTR	"s,t"	$	con las letras iniciales 's' y 't' se declaran como cadenas de caracteres o string.

El editor reemplaza automáticamente la instrucción de DEFSNG o DEFDBL por DEFFLT.

Normalmente es útil realizar tales declaraciones globales al comienzo del programa. Si uno en alguna parte del programa se aparta de la definición fijada con DEFxxx, solo se debe indicar el sufijo deseado detrás de la variable en cuestión. Esta indicación explícita del tipo de variable siempre tiene prioridad sobre las definiciones globales.La declaración de variables tiene validez hasta que se lleva a cabo otra declaración. El tipo de variable preseleccionado es un valor de punto flotante de 8 byte.

El comando DEFLIST regula la visualización del tipo de variables y las mayúsculas y minúsculas en el listado.

DEFLIST n

n: iexp

DEFLIST fija el formato del listado del programa. La expresión numérica n puede tener un valor entre 0 y 3 (inclusive). En la siguiente tabla se representa el efecto del sentencia DEFLIST sobre la forma de escribir el nombre de la sentencia y de las variables:

n	Sentencia	Variable
0	PRINT	abc
1	Print	Abc
2	PRINT	abc#
3	Print	Abc#

El modo preseleccionado es DEFLIST 0.

DEFLIST 0

Las sentencias y funciones de GFA-BASIC se representan en letras mayúscula. Los nombres de variables, procedimientos y funciones se escriben en minúscula.

DEFLIST 1

Tanto las sentencias de GFA-BASIC y funciones como también los nombres de variables, procedimientos y funciones se representan con la letra inicial en mayúscula, el resto del nombre se escribe en minúscula.

DEFLIST 2

Igual que DEFLIST 0, pero al nombre de todas las variables se agrega adicionalmente un símbolo característico (sufijo).

DEFLIST 3

Igual que DEFLIST 1, pero en el nombre de todas las variables se agrega adicionalmente un símbolo característico (sufijo).

$ text

text: Secuencia de caracteres cualesquiera

La sentencia $, que el intérprete trata igual que un comentario REM, sirve para el manejo del compilador. Una descripción exacta se encuentra en las instrucciones de la versión del compilador para el GFA-BASIC 3.0.

El editor

Fundamentos

El editor del GFA-BASIC 3.0 no es un editor de texto convencional, sino que fue creado especialmente para el desarrollo de programas. Por ejemplo, durante la creación del programa ya se reconocen sentencias sintácticamente erróneas. Además, se ubican automáticamente los comandos dentro de estructuras de repetición (loops) o con instrucciones condicionadas y las abreviaturas de comandos se transfieren al nombre completo del comando (por ej. p en PRINT).

Al escribir un listado de programa siempre se realiza un control de sintaxis, cuando se debe abandonar una línea. Si la sentencia en esta línea no es correcta desde el punto de vista sintáctico , entonces aparece en la segunda línea el mensaje "Syntax Error". Fijando el símbolo de comentario (') al comienzo de la línea, ésta se transforma en comentario. Después se puede abandonar la línea.

En cada línea del programa sólo se puede ingresar una sentencia. Detrás de esta sentencia puede aparecer además un comentario. Esta sentencia no puede tener más de 255 caracteres. Cuando una línea supera los 80 caracteres, su comienzo desaparece en el flanco izquierdo de la pantalla. Por lo tanto, sólo se hace el "scrolling" de esta única línea.

Al abandonar una línea de sentencia, se realiza un control de sintaxis (ver más arriba) y se da formato a la línea. O sea, por ej. hay que eliminar espacios en blanco superfluos (por ej. de 2 + 2 resulta entonces 2+2) y escribir las palabras de las sentencias y los nombres de las variables según el DEFLIST conectado. Si se preseleccionó el DEFLIST 0, por ej., todas las palabras de las sentencias se escriben en mayúscula y todos los nombres de las variables en minúscula.

El bloque de teclas del cursor

El cursor se puede dirigir a través del bloque de teclas del cursor. La instalación de las teclas es:

Flecha hacia la izquierda	:	el cursor se desplaza un carácter a la izquierda
Flecha hacia la derecha	:	el cursor se desplaza un carácter a la derecha
Flecha hacia arriba	:	el cursor se desplaza una línea hacia arriba
Flecha hacia abajo	:	el cursor se desplaza una línea hacia abajo

Los movimientos del cursor están sometidos a algunas limitaciones. Como máximo se puede ubicar un carácter detrás del último carácter de una línea y una línea por debajo de la última línea del programa.
Cuando el cursor pasa a una línea, que es más corta que la posición actual de columna del cursor, el cursor salta al final de esta línea, en lo demás mantiene su posición actual de columna (a diferencia de los editores de versiones más antiguas de GFA-BASIC).

También es posible ubicar el cursor con el ratón. Para ello se señala con la flecha del ratón la posición deseada y se oprime la tecla

izquierda del ratón.

Al pulsar la tecla de Insert se introduce una línea en blanco entre la línea sobre el cursor y la línea del cursor, si no se ha realizado ningún cambio en la línea actual. El cursor se ubica al comienzo de esta línea en blanco. Con Clr-Home se lleva el cursor al márgen superior izquierdo por debajo de la zona del menú , con Control+Clr/Home se lleva el cursor al comienzo del listado del programa.

Con ayuda de la tecla Undo se pueden revertir cambios en una línea del programa, en la medida que no se ha abandonado todavía la línea. La tecla Help posibilita "desdoblar y doblar" procedimientos en el listado del programa.

Quiere decir: cuando se ubica el cursor en una línea, en la cual aparece el comando PROCEDURE y luego se pulsa la tecla Help, entonces desaparecen todas las líneas de sentencia en el listado hasta el siguiente RETURN (inclusive). En lugar de eso se coloca una flecha y un espacio en blanco '> ' delante del comando PROCEDURE.

En esta situación tampoco se pueden realizar cambios en los nombres del procedimiento o la lista de parámetros del subprograma. Para poder volver a ver las líneas de sentencia entre PROCEDURE y RETURN, se desplaza el cursor a la línea , que comienza con la flecha y se pulsa nuevamente la tecla Help o se suprime el carácter '>'.

Este " Doblar" (Folding) de un subprograma hace posible crear listados breves y claros, en los cuales solo se "desdobla" el subprograma, con el cual se trabaja en ese momento. Un programa con procedimientos doblados puede ser así, por ej.:

```
init
main_menu
'
> PROCEDURE INIT
> PROCEDURE menu_principal
> PROCEDURE zona de menú
> PROCEDURE cargar
> PROCEDURE almacenar
> PROCEDURE elaborar
> PROCEDURE buscar información
> PROCEDURE mostrar información
```

Con un procedimiento desdoblado el programa es entonces el siguiente:

```
init
main_menu
'
> PROCEDURE init
> PROCEDURE menu_principal
> PROCEDURE zona de menú
PROCEDURE cargar
  FILESELECT "*.RSC","",rsc_fichero$
  IF NOT EXIST(rsc_fichero$)
     ALERT 1,"No existe fichero !",1,"Interrupción",r%
  ELSE
     IF RSRC-LOAD(rsc_fichero$)=0
       ALERT 1, "Error en la carga del fichero !",1, "Interrupción",r%
```

```
      END
    ENDIF
  ENDIF
RETURN
> PROCEDURE almacenar
> PROCEDURE elaborar
> PROCEDURE buscar información
> PROCEDURE mostrar información
```

El teclado numérico

El teclado numérico sirve normalmente para ingresar números y algunos otros caracteres. Pero también se puede usar en combinación con la tecla de control. Su instalación corresponde entonces a la instalación en teclados con NUMLOCK conectado (por ej. en PC's). La instalación es:

Control y 4 Desplaza cursor un carácter a la izquierda
Control y 6 Desplaza cursor un carácter a la derecha
Control y 8 Desplaza cursor una línea hacia arriba
Control y 2 Desplaza cursor una línea hacia abajo
Control y 7 Salto al comienzo del programa
Control y 1 Salto al final del programa
Control y 9 Paso a la página anterior
Control y 3 Paso a la página siguiente
Control y 0 Corresponde a Insert
Control y . Corresponde a Delete.

El teclado numérico también puede trabajar en un modo, en el cual éstas teclas son efectivas sin Control. La conmutación se hace a través de otras (combinaciones) de las teclas de control del teclado numérico.

Control y - , así como Control y (corresponden a NUMLOCK. Entonces al pulsar una tecla del teclado numérico resulta el comando indicado anteriormente con Control. El modo NUMLOCK se puede volver a anular si se vuelve a pulsar la misma tecla.

Otras teclas editoras en el teclado principal

Con la tecla DELETE se puede suprimir el carácter que está en la posición del cursor; el resto de la línea se acopla sin dejar espacios en blanco. La tecla Backspace suprime el carácter, que se encuentra a la izquierda del cursor y acopla el resto de la línea sin dejar espacios en blanco.

Al pulsar la tecla tab (tabulador) el cursor salta a la siguiente posición de tabulación a la derecha. Estas posiciones se encuentran en intervalos de ocho caracteres. Control + tab lleva el cursor a la siguinete posición de tabulación a la izquierda.

Las teclas Return y Enter hacen saltar el cursor al comienzo de la siguiente línea. Pulsando la tecla Escape, se pasa al modo directo.

Comandos de control

En los párrafos anteriores ya se han mencionado muchos comandos de Control. Aquí se han de resumir una vez más estos comandos y los comandos de las teclas de Control no mencionadas allí (excepto los comandos de Control, que trabajan junto con el teclado numérico):

Control + Delete suprime la línea en la que está el cursor
Control + U (undelete)

inserta la línea que se suprimió como última con Control + Delet o Control + Y. Esto sirve, por un lado, para restaurar líneas, que se han suprimido sin querer. Por otro lado, así se pueden desplazar o copiar fácilmente líneas individuales (por ej. para copiar: sostener la tecla de Control y pulsar Delete y dos veces U).

Control+Y	Suprime la línea que está en la posición del cursor
Control+N	Inserta una línea en blanco por encima de la línea de órden que está en la posición del cursor (como Insert), también cuando se han llevado a cabo modificaciones en la línea.
Control+Q	Llama al menú de bloque (igual que la tecla de función F4).
Control+B	Marca el comienzo del bloque (bloque).
Control+K	Marca el fin del bloque.
Control+R	Pasa a la página anterior.
Control+C	Pasa a la página siguiente.
Control+E	Sustituye texto
Shift+Control+E	Sustituye preguntando con el string de buscar y sustituir
Control+F	(find), busca texto.
Shift+Control+F	Busca texto preguntando con el string de buscar.
Control+cursor izquierda	Salta al comienzo de la línea.
Control+cursor derecha	Salta al final de la línea.
Control+cursor arriba	Pasa a la página anterior.
Control+cursor abajo	Pasa a la página siguiente.
Control+Clr+Home	Salta al principio del programa
Control+Z	Salta al final del listado del programa.
Control+Tab	Salta a la posición de tabulación a la izquierda.
Control+G (goto)	Abre el indicador de número de línea para ingresar un número de línea (se pasa entonces a la correspondiente línea).

Un grupo especial de Comandos de control hace posible fijar y acceder a marcas en el editor. Estas marcas valen solo para el editor, o sea no tienen nada que ver con las marcas que utiliza GOTO o RESTORE. Estas marcas del editor se pueden ubicar en la posición del cursor pulsando Control y un Número del teclado principal. A continuación se puede saltar a la marca deseada oprimiendo Alternate y este número.

Las combinaciones de teclas de Alternate con los números 7-9 y 0 están preconfiguradas. Al pulsar Alternate y 7 se salta a la última posición del cursor antes de pasar al modo directo o antes del último comienzo de programa. Alternate y 8 salta a la posición, en la que se encontraba el cursor al comienzo del editor. Con Alternate y 0 se salta a la última posición del cursor, en que se realizó una modificación.. Alternate y 9 salta a la posición, en la que se inició la última operación de búsqueda.

La zona de menú y las teclas de función

En las dos líneas superiores de la pantalla del editor se encuentran dos líneas del menú. Bien a la izquierda aparece el símbolo ATARI, que cuando se activa desdobla un menú Pull-Down. Este contiene los títulos del menú del símbolo ATARI y el GFA-BASIC.

Cuando se activa el item del menú GFA-BASIC3.00 se visualizan los números de versión con las dos posibilidades de elección: editor y menú. Si se activa el botón del editor, se vuelve al editor GFA-BASIC. Al elegir el menú se vuelve al menú Pull-Down.

El título del menú GFA-BASIC contiene los siguientes items de menú:

Save — Salta al editor, luego se visualiza un File-Select-Box, y se puede ingresar un nombre de fichero, bajo el cual se ha de guardar el programa actual.

Load — Salta al editor, después de lo cual se puede ingresar el nombre de un fichero en el File-Select-Box que se visualiza.

Deflist — Posibilita la conexión del modo Deflist a través de un Alert-Box (comparar apartado Casos especiales).

Nombres nuevos — Con ayuda de este item del menú se puede conectar un modo, en el cual se debe confirmar el ingreso de nuevas variables en el programa actual. Este modo está desconectado en la preselección. Cuando se emplea una variable que no se ha ingresado aún o también un nombre de un procedimiento o de una función y este modo está activo, aparece un Alertbox, en el cual se puede indicar si se quiere incorporar esta nueva variable al programa o si se trata de un error de escritura (en este caso se visualiza el mensaje de error "Syntax Error").

En el flanco derecho de la zona de menú se encuentra en la parte superior un reloj y debajo de este la indicación del número de línea de texto actual. El manejo de estos dos elementos de la zona de menú se explicará más adelante.

Entre el símbolo ATARI y el reloj se visualizan veinte palabras de comando, de a dos una sobre la otra, las cuales se pueden activar con el ratón o a través de las teclas de función. La línea inferior de las palabras de comando se puede activar pulsando una tecla de función; los comandos de la línea superior pulsando la tecla Shift y la correspondiente tecla de función. A la palabra de comando en la parte inferior izquierda pertenece por ej. la tecla F1, la palabra de comando por encima de ésta (Save) se llama con Shift+F1.

Debajo del símbolo ATARI hay lugar para otros dos caracteres. Allí se visualiza el estatus de la tecla CAPSLOCK y NUMLOCK. Si se pulsa la tecla CAPSLOCK se visualiza a la izquierda debajo del símbolo ATARI una flecha que apunta hacia arriba. Cuando se pulsa la tecla NUMLOCK, o sea Control+(, se visualiza a la derecha debajo del símbolo ATARI el símbolo de elevación a potencia ^. El modo NUMLOCK y CAPS-LOCK también se pueden seleccionar activando la tecla del ratón izquierda debajo del símbolo ATARI.

Los items del menú tienen el siguiente significado:

Load F1

El comando Load hace posible cargar un programa GFA-BASIC3.0. El formato empleado aquí emplea los llamados tokens de sentencia. Un fichero de programa de este tipo se puede cargar y almacenar con gran velocidad . Para este formato se usa normalmente el sufijo de fichero .GFA.

Las versiones más antiguas de GFA-BASIC usan otro formato token. Por eso,sus programas en la versión antigua se deben almacenar con Save,A y cargar en GFA-BASIC con Merge. Luego se pueden almacenar en 3.0 con Save y cargar con Load.

Save Shift+F1

Se visualiza un File-Select-Box, en el cual se puede indicar un nombre. Bajo este nombre se puede almacenar entonces el programa, que se encuentra en ese momento en el editor. Aquí se usa el formato tokenizado bajo Load. El sufijo del fichero conectado previamente se llama .GFA. Se agrega automáticamente al nombre del fichero, si el usuario no indica sufijo.

Si ya existe un fichero con el nombre ingresado, éste se almacena con el mismo nombre de fichero con el sufijo .BAK.

Merge F2

Con este comando se puede insertar en el programa actual un fichero en el formato ASCII. La inserción se realiza antes de la línea en que está el cursor. El sufijo del fichero conectado previamente es .LST. Los programas de versiones más antiguas de GFA-BASIC se almacenen de acuerdo con la versión antigua con Save,A y se vuelven a cargar en GFA-BASIC con Merge.

* * > Las líneas del programa con este sufijo surgen con el Merge * * > de un programa, que el Intérprete no entiende.

Save,A Shift+F2

Con Save,A se puede almacenar el programa actual en formato ASCII. Un fichero almacenado en este formato se puede volver a cargar con Merge. El sufijo del fichero conectado previamente es .LST. Se agrega automáticamente, si el usuario no ingresa ningún sufijo.
Si ya existe un fichero bajo el nombre ingresado, éste se almacena bajo el mismo nombre con el sufijo .BAK.
Llist F3

Este comando inicia la impresión del programa que se encuentra en el editor. El formateo de la impresión se puede influenciar con las llamadas instrucciones de punto. Estas instrucciones se insertan en el listado del programa igual que líneas de sentencia normales. Son (x es una anotación que se usa para indicar cuántos números aparecerán):

.ll xx - longitud máxima de la línea
.pl xx - longitud máxima de la página
- pasar a la siguiente página

```
                - conditional page
                - enumeración de las páginas
                - márgen izquierdo
.ff xxx         - cadena de caracteres form-feed ( para impresoras que
                  tienen otros valores de form-feed, se puede ingresar
                  una cadena de sustitución. Previamente se conectó
                  .ff\012)
.he cabecera    - texto de la línea de cabecera
.fo pié         - texto de la línea de pie de página
.lr xx          - márgen izquierdo
.l-             - Con esta indicación se logra que las líneas
                  siguientes no aparezcan en el listado. El listado se
                  puede volver a conectar mediante un .l+ que aparece
                  más adelante en el programa.
.l+             - ver .l-
.n1 a .n9       - conecta la enumeración de la líneas con 1 a 9
                  lugares.
.n0             - desconecta la enumeración de las líneas.
```

En texto de la línea de cabecera y al pie de página así como en el string FF se pueden insertar otras anotaciones que indican cuántos números aparecerán.

```
\xxx            - Carácter con el ASCII-Code xxx
\d              - Fecha
\t              - hora
#               - número de página
```

Para poder evidenciar los caracteres especiales # y \, hay que fijar otro carácter \ delante. Por lo tanto, la combinación de caracteres \\ evidencian un \ en la impresora y la combinación \# un #.

Quit Shift+F3

Hace posible abandonar intérpretes GFA-BASIC.

Block F4

Si no se marcó ningún bloque y se selecciona éste item del menú, entonces se visualiza en la zona de menú el texto "Block???", para indicar, que la elección del comando bloque no tiene sentido en esta situación. Pero si se marcó un bloque, se visualiza un menú en la línea superior de la pantalla. Los items individuales de este menú se pueden activar con el ratón o pulsando una letra de comando. Los items del menú son (Entre paréntesis aparece siempre la tecla, con la que se puede llamar el item del menú):

Copy Copia el bloque en la posición actual del cursor. En este caso queda marcado el bloque.

Move Desplaza el bloque a la posición actual del cursor. En este caso se elimina la marca del bloque.

Write Almacena el bloque como fichero ASCII.

Llist Imprime el bloque.

Start Salta al comienzo del bloque.

End Salta al final del bloque.

^Del Suprime el bloque (Control-D).

Hide: Elimina la marca del bloque.

Pulsando otra tecla (o la tecla del ratón) fuera de esta zona se elimina el menú del bloque.

New Shift+F4

Suprime el programa que se encuentra en el editor.

BlkEnd F5

Con BlkEnd se puede marcar la línea en que está el cursor como final de bloque. Si la marca del comienzo del bloque se encuentra sobre esta línea, entonces el bloque se enmascara con una matriz de puntos en la versión monocromática. En la versión color se representa el bloque marcado en otro color. Este comando también se puede ejecutar pulsando Control+K.

BlkSta Shift+F5

Con BlkSta se marca la línea en que está el cursor como comienzo del bloque. Si detrás de esta línea se fijó anteriormente un final de bloque, la zona del bloque se enmascara con una matriz de puntos. Este comando también se puede ejecutar pulsando Control+B.

Find F6

Después de inicializar el comando Find se puede ingresar un texto a buscar . La búsqueda comienza en la línea en que está el cursor. La búsqueda se puede volver a llamar pulsando Control+F o Control+L, sin que se visualice la pregunta del texto a buscar.

Cuando se encontró el texto buscado, el cursor aparece al comienzo de la línea que contiene el texto buscado; cuando no se encontró aparece el cursor al final del programa.

Cuando se vuelve a llamar el comando Find, entonces aparece en la línea, en la que se puede ingresar el string de buscar, el último string de buscar usado. Al ingresar el texto a buscar se dispone de las siguientes posibilidades de edición:

Cursor izquierda	El cursor se desplaza un carácter a la izquierda (en la medida en que no se encuentra bien a la izquierda).
- derecha	El cursor se desplaza un carácter a la derecha (en la medida en que no se encuentra al final de la cadena de búsqueda).
Delete	Suprime el carácter de la cadena de búsqueda, que está debajo del cursor, el resto de la cadena de búsqueda se acopla de derecha a izquierda.

Backspace	Suprime el carácter a la izquierda del cursor (en la medida en que no se encuentra ya en el márgen izquierdo) y envuelve el resto de la cadena de búsqueda.
Escape	Suprime el campo de entrada
Return/Enter	Confirma el texto de búsqueda ingresado y comienza a buscar.

El comando Find también se puede llamar con la combinación de teclas Shift+Control+F o Shift+Control +L. En procedimientos "doblados" no se encuentra texto (ver tecla Help en el párrafo teclado del cursor).

Replace Shift+F6

Este comando sirve para sustituir un texto por otro. Después de llamarlo, se pregunta por el texto que se quiere sustituir. Luego se pregunta, cual es el texto que lo sustituirá. Entonces comienza en el programa la búsqueda del texto que se quiere sustituir desde la posición del cursor. Cuando se encuentra el texto buscado, el cursor salta al comienzo de la línea correspondiente.

Entonces se puede realizar la sustitución pulsando la combinación de teclas Control+E. Si se sigue pulsando la combinación de teclas se busca la siguiente parte del texto a sustituir o, si se encuentra en la posición actual del cursor, la sustituye.

El comando Replace también se puede llamar con la combinación de teclas Shift+Control+E. Las posibilidades de edición, de las que se dispone al ingresar el texto de búsqueda o de sustitución, se han señalado en la descripción del comando Find. En procedimientos "doblados" no se encuentra ningún texto (ver tecla Help en el párrafo teclado del cursor).

Pg Down F7

"Da vuelta la página" del texto del programa "enrollando" el texto de la pantalla hacia abajo. También se puede llamar con Control+C.

Pg Up Shift+F7

"Da vuelta la página" del texto del programa, "enrollando" el texto de la pantalla hacia arriba. También se puede llamar con Control+R.

Insert/Overwr F8

Pasa del modo INSERT (inserción de texto) a modo OVERWRITE (cada nuevo carácter es tecleado sobre el carácter que está en la posición del cursor, los caracteres sobre los que se teclea son suprimidos) y viceversa.

Txt16/Text8 Shift+F8

Este comando solo está a disposición en la definicion monocromatica. En el modo 16 se representan letras con un alto de 16 pixel, o sea que en la pantalla se visualizan simultáneamente 23 líneas. En el modo 8 el alto de las letras es de solo 8 pixel, pero en la pantalla se visualizan simultáneamente 48 líneas. Con Txt16/Text8 se puede conmutar

entre estos dos modos.

Flip F9

Con este comando se puede conmutar a la pantalla de salida, donde se visualiza la situación que dejó el último programa llamado. Pulsando una tecla (también la tecla del ratón) se puede volver a la pantalla del editor.

Direct Shift+F9

Con este comando accede al modo directo. De esta manera puede ingresar comandos de GFA-BASIC para que se ejecuten inmediatamente después de pulsar la tecla Return o Enter. Algunos comandos, por ej. comandos de estructuras de repetición (loops), no se pueden llamar en el modo directo. A este modo también se puede acceder pulsando la tecla Escape.

En el modo directo además es posible volver a llamar las últimas ocho sentencias (que se ingresaron en el modo directo) subiendo y bajando el cursor con las teclas.

Pulsando la tecla Undo se visualiza la última sentencia. Con las combinaciones de teclas Control+Cursor izquierda o bien Control+Cursor derecha se pueden alcanzar el comienzo y final de la línea. Además, pulsando la tecla Insert se puede conmutar entre el modo de inserción y normal (por defecto).

El modo directo se puede volver a abandonar pulsando la tecla Escape seguida de Return. En forma alternativa también se puede oprimir simultáneamente Control+Shift+Alternate.

Para llamar instrucciones de varias líneas, se puede escribir en el editor un procedimiento, que contiene estas sentencias. En el modo directo se puede llamar entonces a este procedimiento.

Test F10

Después de la ejecución de esta instrucción se examina si todas las estructuras de repetición (loops), los subprogramas y las instrucciones condicionadas del programa actual cierran. O sea, se lleva a cabo una prueba de estructura.

Con Control, la tecla Shift izquierda y Alternate se puede interrumpir un programa en ejecución.

Run Shift+F10

Inicia la ejecución del programa que se halla en el editor. Si este programa contiene un error estructural, por ej. un loop FOR-NEXT no cerrado, se visualiza el correspondiente mensaje de error y no se inicia la ejecución del programa.

Control+Shift+Alternate

Con la combinación de teclas Control+tecla Shift izquierda+Alternate se puede interrumpir el programa en ejecución.

El indicador de línea y de la hora

En la parte inferior derecha de la zona de menú se encuentra el

indicador de la línea en que se encuentra el cursor. En este indicador de línea también se puede incorporar un número de línea, de tal manera que el cursor salta a la línea correspondiente. Con este fin se puede activar la zona de indicación mediante el ratón o se puede inicializar con el comando Control+G.

En este campo solo se pueden ingresar números. Las posibilidades de edición para el ingreso de los números de línea deseados son:

Cursor izquierda	El cursor se desplaza un carácter a la izquierda (en la medida en que no se encuentra bien a la izquierda).
- derecha	El cursor se desplaza un carácter a la derecha (en la medida en que no se encuentra bien a la derecha).
Backspace	Tiene la misma función que Cursor izquierda.
Escape	Suprime el campo de entrada.
Return/Enter	Confirma el número ingresado.

Sobre el indicador del número de línea se encuentra un reloj, que indica la hora del sistema. Después de activar el reloj con la flecha del ratón, existe la posibilidad de volver a ajustar el reloj. Las posibilidades de edición para el ingreso de la hora son las mismas que para el campo indicador del número de línea. Solo la tecla Escape tiene aquí otro significado, ya que interrumpe el ingreso manteniendo para la hora el valor viejo.

Casos especiales

```
DEFBITf$
DEFBYTf$
DEFINTf$
DEFWRDf$
DEFFLTf$
DEFSTRf$
```

f$: constante string o alfanumérica

La instrucción DEFxxx sirve para declaración simplificada de variables. En este caso xxx es una anotación que indica el número de abreviaturas de tipos de variables, que se enumeran y explican en la siguiente tabla:

DEFxxx	f$	Sufijo	Todas las variables...
DEFBIT	"B"	!	con la letra inicial 'b' se declaran variables booleanas.
DEFBYT	"by"	I	con las letras iniciales 'by' se declaran como entera de un byte.
DEFWRD	"w"	&	con la letra inicial 'w' se declaran como entera de dos byte con signo.

DEFINIT	"i-k,m-p"	%	con las letras iniciales i' a 'k'y 'm' a 'p' se declaran como enteras de 4 byte con signo.
DEFFLT	"x-z"	#	Con las letras iniciales 'x' a 'z'se declaran como valores de punto flotante de 8 byte.
DEFSTR	"s,t"	$	con las letras iniciales 's' y 't' se declaran como cadenas de caracteres o string.

El editor reemplaza automáticamente la instrucción de DEFSNG o DEFDBL por DEFFLT.

Normalmente es útil realizar tales declaraciones globales al comienzo del programa. Si uno en alguna parte del programa se aparta de la definición fijada con DEFxxx, solo se debe indicar el sufijo deseado detrás de la variable en cuestión. Esta indicación explícita del tipo de variable siempre tiene prioridad sobre las definiciones globales.La declaración de variables tiene validez hasta que se lleva a cabo otra declaración. El tipo de variable preseleccionado es un valor de punto flotante de 8 byte.

El comando DEFLIST regula la visualización del tipo de variables y las mayúsculas y minúsculas en el listado.

DEFLIST n

n: iexp

DEFLIST fija el formato del listado del programa. La expresión numérica n puede tener un valor entre 0 y 3 (inclusive). En la siguiente tabla se representa el efecto del sentencia DEFLIST sobre la forma de escribir el nombre de la sentencia y de las variables:

n	Sentencia	Variable
0	PRINT	abc
1	Print	Abc
2	PRINT	abc#
3	Print	Abc#

El modo preseleccionado es DEFLIST 0.

DEFLIST 0

Las sentencias y funciones de GFA-BASIC se representan en letras mayúscula. Los nombres de variables, procedimientos y funciones se escriben en minúscula.

DEFLIST 1

Tanto las sentencias de GFA-BASIC y funciones como también los nombres de variables, procedimientos y funciones se representan con la letra inicial en mayúscula, el resto del nombre se escribe en minúscula.

DEFLIST 2

Igual que DEFLIST 0, pero al nombre de todas las variables se agrega adicionalmente un símbolo característico (sufijo).

DEFLIST 3

Igual que DEFLIST 1, pero en el nombre de todas las variables se agrega adicionalmente un símbolo característico (sufijo).

$ text

text: Secuencia de caracteres cualesquiera

La sentencia $, que el intérprete trata igual que un comentario REM, sirve para el manejo del compilador. Una descripción exacta se encuentra en las instrucciones de la versión del compilador para el GFA-BASIC 3.0.

2 Variables y administración de memoria

Tipos de variables

El GFA-BASIC 3.0 dispone de los siguientes tipos de variables:

Nombre	Sufijo	Requerimiento de lugar de almacenamiento
Booleana	!	1 byte (en arreglos un bit)
Byte	I	1 byte
Palabra	&	2 byte
Entero	%	4 byte
Decimal	#	8 byte
Cadena	$	(depende de la longitud de la cadena)

Las variables booleanas solo pueden tener los valores 0 (falso) o 1 (verdadero). Si se les asigna un valor distinto de cero, este valor se interpreta como -1. Estas variables son provistas del sufijo ! y ocupan un espacio de memoria de 1 byte. En arreglos booleanos, un elemento del arreglo requiere de un bit.

```
Ejemplos: b!=TRUE
c!=x>y
```

El tipo de variable Byte puede asumir valores entre 0 y 255. Valores superiores para 'x' se tratan como excesivos. El sufijo de estas variables es la línea vertical I. Como ya lo dice su nombre, este tipo de variable ocupa un byte.

```
Ejemplo: xI=128
```

Word es un tipo de variable de contenido entero de 2 bytes con signo. El sufijo de este tipo es &.

El rango representable de números alcanza de -32768 a 32767.

```
Ejemplo: x&=32767
```

Entero es un tipo de variable de contenido entero de 4 bytes con signo. Como sufijo se usa el símbolo %.

El rango representable de números alcanza desde -2147483648 a 2147483647.

```
Ejemplo: x%=2000000000
```

Decimal es un tipo de variable de punto flotante que ocupa 8 bytes en la memoria. En este tipo no se usa sufijo o el símbolo #.El rango representable de números alcanza desde 2.225073858507E -308 a 3.595386269725E +308.

Cadenas de caracteres (strings) se caracterizan con el sufijo $. Su longitud máxima es de 32767 caracteres. Las cadenas de caracteres se administran con el llamado descriptor. Un descriptor de este tipo tiene una longitud de seis bytes. Los primeros cuatro bytes contienen la dirección de la cadena de caracteres, los últimos dos bytes contienen la longitud del string. Detrás del string se inserta un byte, para el caso en que la longitud del string sea desigual. Luego aparece la dirección del descriptor (backtrailer).

Con ayuda de las funciones VARPTR (o V) y ARRPTR (o *) se pueden obtener las direcciones de todas las variables. En strings se obtiene con VARPTR la dirección del primer byte de la cadena de caracteres, con ARRPTR se obtiene la dirección del descriptor del string.

Mediante las funciones VARPTR (o V) y ARRPTR (o *) se pueden obtener las direcciones de todos los tipos de variables. En strings se obtiene con VARPTR la dirección del primer byte de la cadena de caracteres, con ARRPTR se obtiene la dirección del descriptor del string.

Cuando se trata de arreglos se pueden obtener las direcciones de los elementos individuales del arreglo con VARPTR/V: (por ej. V:x%(5)). Con ayuda de ARRPTR/* se obtiene la dirección del descriptor del arreglo (por ej. ARRPTR (x%())). Con VARPTR/V: se puede obtener entonces la dirección de elementos individuales del campo, mientras que ARRPTR determina la dirección del descriptor del campo.

Campos

DIM,DIM?
OPTION BASE
ARRAYFILL

Con todos los tipos de variables se pueden producir campos. La instalación de uno de estos campos se logra con la instrucción DIM. La función de pregunta DIM? indica, cuántos elementos contiene el arreglo.

La administración de los arreglos en la memoria ocurre a través de descriptores. Un descriptor de este tipo es una estructura de seis bytes de longitud, en cuyos primeros cuatro bytes está contenida la dirección del campo. Los siguientes dos bytes indican cuántas dimensiones tiene el campo. El campo comienza cada vez con cuatro bytes para cada dimensión del arreglo, comenzándose con la última dimensión. Luego siguen los contenidos de los elementos del campo. En las variables string éstos son los descriptores de cadenas de caracteres.

Por ej. después de DIMa%(2,3) resulta la siguiente situación; a través de *a%() se obtiene la dirección del descriptor del campo. El número de dimensiones del arreglo figura en los últimos dos de los seis bytes del descriptor, o sea INT{*a%()+4} da el valor 2. El arreglo comienza con la cantidad de subdivisiones de la segunda dimensión, donde se cuenta también el elemento cero (suponiendo que vale OPTION BASE 0). Luego figura la cantidad de subdivisiones de la primera dimensión. Entonces

{{*a%()}} da el valor 4 y
{{*a%()}+4} da el valor 3.

A esto se acoplan los 12 elementos del arreglo, requiriendo cada vez 4 bytes de espacio de memoria en la secuencia.

a%(0,0) a%(1,0) a%(2,0) a%(0,1) a%(1,1) a%(2,1) etc.

DIMx(d1,[d2,...])[,y(d1,[d2,...])]
DIM?(x())

x,y: nombre de la variable (cualquier tipo de variable)
d1,d2:iexp

Con la instrucción DIM se pueden instalar campos numéricos y de cadenas de caracteres. La estructura de un arreglo de este tipo se ha descripto en la introducción de este apartado.

Desde el punto de vista práctico la cantidad posible de dimensiones del arreglo no es limitada.La cantidad de elementos de arreglos en arreglos multidimensionales es limitada en el sentido en que la última dimensión debe ser menor que 65535. Además, el producto de la cantidad de elementos de campo debe ser menor que 65535. (Así, por ej., se permite DIMa%(100,10,10), ya que la última dimensión (10) y el producto de la cantidad de elementos del campo (100*10*10=10000) es menor que 65535).

Con ayuda de OPTION BASE se puede fijar, si hay un elemento del arreglo igual a cero. En un arreglo sólo pueden existir variables del mismo tipo. El acceso a los elementos del arreglo ocurre a través de sus índices. La función DIM? calcula cuántos elementos tiene el arreglo.

Ejemplo:

```
DIM x(10)
x(4)=3
PRINT x(LEN("Test"))
PRINT DIM?(x())
DIM y%(2,3)
PRINT DIM?(y%())
```

--> Instala dos arreglos. Escribe en uno de los arreglos un valor y lo vuelve a leer. Imprime en pantalla la cantidad de elementos de ambos arreglos (11 y 12).

```
OPTION BASE0
OPTION BASE1
```

Con la instrucción OPTION BASE se puede determinar, si en arreglos ha de existir un elemento igual a cero o no. Con OPTION BASE0 se puede instalar un elemento igual a cero de este tipo, con OPTION BASE 1 no. Los índices del arreglo comienzan en este caso con 1.

Con la instrucción OPTION BASE no se altera el contenido de los campos, aunque en determinados casos varía el índice de los elementos individuales por el valor 1.

Cuando no se da ninguna instrucción OPTION BASE, existen elementos del arreglo con el índice 0.

Ejemplo

```
DIM x%(3)
FOR i%=3 DOWNTO 0
  x%(i%)=i%
  PRINT i%,x%(i%)
NEXT i%
OPTION BASE 1
FOR i%=3 DOWNTO 0
  PRINT i%,x%(i%)
NEXT i%
```

--> Escribe los índices y el contenido del arreglo x%() sobre la pantalla. El programa finaliza con el mensaje de error índice del campo erróneo, ya que x%() no posee ningún elemento igual a cero según OPTION BASE.

```
ARRAYFILLx(),y
```

x: nombre de un arreglo con tipo de variable numérica
y: aexp

La instrucción ARRAYFILL iguala todos los elementos del arreglo 'x' al valor de la expresión numérica 'y'.

Ejemplo

```
DIM x(10)
PRINT x(4)
ARRAYFILL x(),5+1
PRINT x(4)
```

--> Evidencia el número cero, ya que en la instrucción DIM todos los elementos de campo se fijan en cero. Después de llenar el arreglo se visualiza el número 6 en la pantalla.

Conversión de tipo

```
TYPE(x)
```

x: iexp

Con ayuda de la función TYPE se puede controlar, si se transfirió un pointer a una variable. Se devuelve un número característico o el valos -1 para el tipo de esta variable. De acuerdo con el tipo de variable que señala 'x', se obtienen los siguientes valores:

Decimal	-->	0
String	-->	1
Entero	-->	2
Booleana	-->	3
Arreglo de decimales	-->	4
Arreglo de strings	-->	5
Arreglo de enteros	-->	6
Arreglo booleano	-->	7
Palabra	-->	8
Byte	-->	9
Arreglo de palabras	-->	12
Arreglo de Bytes	-->	13

Ejemplo

```
a$="test"
x%=4
DIM y(3)
PRINT TYPE(*a$),TYPE(*y())
```

--> Imprime en pantalla los números 1 y 4.

```
ASC(a$)
CHR(x)
```

```
a$:  sexp
x:   aexp
```

ASC y CHR son funciones inversas.

La función ASC da el código ASCII del primer carácter en el string a$. En el caso en que a$ es un string nulo, se imprimirá un cero como mensaje.

CHR$ da el carácter de ASCII, que tiene el código 'x'. Aquí solo se valora el lowbyte de 'x' (corresponde al valor x AD255).

Ejemplo

```
PRINT ASC("Prueba")
code|=ASC(CHR$(65))  ! CHR$(65) equivale a 'A'
PRINT code|,CHR$(189)
```

--> Imprime los números 84,65 y el símbolo Copyright en la pantalla.

Las funciones SR$, BI$, C$ y H$ convierten una expresión numérica 'x' en una cadena de caracteres. En esta cadena de caracteres figura un número, que se calcula a partir de la expresión numérica 'x'.

En 'y' usted dice qué longitud debe tener la cadena de caracteres devuelta. Si 'y' es mayor que la cantidad de caracteres necesarios para la representación del número, entonces se completa adelante el string en STR$ con espacios en blanco. En las funciones BIN$, OCT$ y HEX$ el string de resultados se completa eventualmente con ceros.

```
STR$(x)
STR$(x,y)
STR$(x,y,z)
```

```
x,y,z:  aexp
```

STR$ convierte un número 'x' en string. Un segundo parámetro 'y' indica, en cuántos lugares a la izquierda se debe completar el string de resultados con espacios en blanco, o sea cuántos caracteres del string se han de emplear (contado a partir del final de la cadena de caracteres).

STR$(x,n) corresponde a RIGHT$(SPACE$(n)+STR$(x),n).

Otra variante de STR$ dispone de un tercer parámetro 'z'. Aquí se formatea el número 'x' en 'y' lugares con 'z' decimales (redondeado).

Ejemplo

```
a=123.4567
PRINT STR$(a,6,2)
PRINT STR$(PI,5,3)
PRINT STR$(PI,2,2)
```

--> Imprime los números 123.46,3.142 y 14 en la pantalla.

```
BIN$(x[,y])
OCT$(x[,y])
```

```
HEX$(x,y)

x,z:  iexp
```

BIN$ convierte un número entero en la representación binaria.

Quiere decir, que el número se presenta después de la conversión en el sistema numérico en base 2. En este caso se representan los números únicamente con las cifras 0 y 1. El parámetro 'y' indica, cuántos lugares (1 a 32) se emplean.

OCT$ convierte un valor entero en la correspondiente representación octal del sistema numérico en base 8. En este caso se representan los números con las cifras 1 a 7. El parámetro opcional 'y' indica, cuántos lugares (1 a 11) se emplean.

HEX$ convierte un número entero en la correspondiente representación hexadecimal. En este caso se trata de un sistema numérico en base 16 con las cifras 0 a 9 y las letras A a Fw> El segundo parámetro opcional 'y' indica, cuántas cifras hexadecimales se representan (como máximo ocho).

HEX$(x,n) corresponde a RIGHT$(STRING$(n,48)+HEX$(x),n).

Ejemplo

```
x=32+15
a$=OCT$(16+7,4)
PRINT HEX$(x),a$,BIN$(1+4+16+64,8)
```

--> Imprime 2F,0027 y 01010101 en la pantalla.

```
VAL(a$)
VAL?(a$)

a$:  sexp
```

VAL() convierte una cadena de caracteres en número. En el caso en que VAL() se topa con un carácter, que no se puede interpretar sás como parte componente de un número, se interrumpe el análisis del string. VAL reporta entonces el número que figura al comienzo del string. Si no encuentra ningún número al comienzo del string reportará cero.

Con la indicación &H(hex) o &(bin) o &(oct) se pueden reconocer números con representación hexadecimal, binaria u octal.

Además, con $ se pueden caracterizar números hexadecimales y con % números binarios.

Con AL() se puede examinar, cuántos lugares de un número se convirtieron con VAL(). VAL?() reporta cero, cuando no se indica un número válido.

Ejemplos

```
x=32+15
a$=OCT$(16+7,4)
PRINT HEX$(x),a$,BIN$(1+4+16+64,8)
```

--> Imprime los números 12345,-123 y 4 en la pantalla.

```
PRINT VAL("&H"+"AF")
```

--> Imprime el número 175.

```
PRINT VAL("$AA")
PRINT VAL("%10101010")
```

--> Imprime dos veces el número 170.

```
CVI(a$)  CVL(a$)  CVS(a$)  CVF(a$)  CVD(a$)
MKI(x)   MKL(x)   MKS(x)   MKF(x)   MKD(x)

a$:  sexp
 x:  aexp
```

Las funciones CVI,CVL,CVS,CVF y CVD convierten cadenas de caracteres en valores numéricos. Contrariamente a lo que ocurre con VAL/STR$ no se realiza una conversión a texto claro, sino que es decisiva la representación interna del string y de los números. Las diferentes funciones CVx tienen los siguientes efectos:

CVI convierte un string de dos bytes en un número entero de 16 bit.
CVL convierte un string de 4 bytes en un número entero de 16 bit.
CVS convierte un string de 4 bytes, que contiene un número en formato compatible en ST-BASIC, en un número de punto flotante de GFA-BASIC.
CVF convierte un string de 6 bytes en GFA-BASIC 1.0 y formato 2.0 en un número de punto flotante (en formato 3.0).
CVD convierte un string de 8 bytes en un número de punto flotante en GFA-BASIC formato 3.0.

MKI$,MKL$,MKS$,MKF$ y MKD$ son las funciones inversas de las funciones CVx; su significado resulta por lo tanto de la tabla recién presentada.

Corresponde en estos casos:

```
MKI$(x%)=CHR$(SHR(x%,8))+CHR(x%)
MKL$(x%)=CHR$(SHR(x%,24))+CHR$(SHR(x%,16))+CHR$(SHR(x%,8))+CHR$(x%)
```

Quiere decir, que el byte de mayor valor figura primero.

Un ejemplo de aplicación es la lectura de formatos de números de otros programas, así como el almacenamiento de números para ahorrar espacio, por ej. con "R"-Files.

Ejemplo

```
a$=MKL$(1000)
PRINT CVL(a$),LEN(a$)
b$=MKD$(100.1)
PRINT CVD(b$),LEN(b$)
```

--> Imprime los números 1000 y 4 así como los números 100.1 y 8 en la pantalla.

```
CINT(x)
```

```
CFLOAT(y)

x:  aexp
y:  iexp
```

La función CINT convierte un número de punto flotante 'x' en un valor entero redondeado. En forma análoga convierte CFLOAT un valor entero 'y' en un número de punto flotante. Estas funciones no se necesitan normalmente y se presentan aquí solo a fines de completar el trabajo. Sin embargo, en el compilador tiene un significado.

Ejemplo

```
a=1.2345
a%=10000
b%=CINT(a)
b=CFLOAT(a%)
PRINT b%,b
```

--> Imprime en la pantalla 1 y 10000.

Operaciones de pointer

```
*
BYTE{},CARD{},INT{},LONG{},{},FLOAT{}
SINGLE{},DOUBLE{},CHAR{}
xPEEK,xPOKE
V:,VARPTR
ARRPTR
ABSOLUTE

*x
```

x: svar o un nombre de arreglo seguido de ()

El signo de multiplicación también puede servir de símbolo de pointer (símbolo señalador). Con *x se obtiene en este caso la dirección, en la cual se encuentra la variable 'x' en la memoria. En el caso de cadenas de caracteres se obtiene con *x la dirección del descriptor del string (ARRPTR(x$)). La expresión *x tiene el mismo significado que ARRPTR(x). Esto también vale para arreglos.

Un significado especial tiene esta variable de aplicación del pointer en la transferencia indirecta de arreglos y variables a subprogramas. Para esto se puede usar mejor la instrucción VAR en versión 3.0.

Ejemplo

```
' transferencia indirecta de arreglos
DIM a(3)
change(*a())
PRINT a(2)
'
PROCEDURE change(ptr%)
  SWAP *ptr%,x()
  ARRAYFILL x(),1
  SWAP *ptr%,x()
RETURN
```

--> Se altera el contenido del arreglo a(), sin que aparezca su nombre en el procedimiento change. Se imprime en pantalla el número 1 (ver también SWAP)

```
O mejor en la versión 3.0:
DIM a(3)
change(a())
PRINT a(2)
'
PROCEDURE change(VAR x())
  ARRAYFILL x(),1
RETURN
```

```
PEEK(x)     DPEEK(x)    LPEEK(x)
POKEx,y     DPOKEx,y    LPOKEx,y
SPOKEx,y    SDPOKEx,y   SLPOKEx,y
```

x,y: iexp

Con la función PEEK y la instrucción POKE y sus variantes se pueden seleccionar lugares de la memoria o escribir valores en la memoria. Las diferentes variantes de las instrucciones:

```
PEEK(x)      Lee un byte en la dirección de memoria 'x'.
DPEEK(x)     Lee dos bytes a partir de la dirección de memoria 'x'.
LPEEK(x)     Lee cuatro bytes a partir de la dirección de memoria.

POKE x,y     Escribe el valor 'y' como un byte en la dirección 'x'.
DPOKE x,y    Escribe 'y' como valor de dos bytes en la dirección 'x'.
LPOKE x,y    Escribe 'y' como valor de 4 bytes en la dirección 'x'.
```

En el empleo de DPEEK, LPEEK DPOKE y LPEEK ha de prestarse atención de indicar solo direcciones pares.

Variantes de las instrucciones POKE trabajan con el modo Supervisor. Así también se puede acceder a direcciones protegidas (por ej. 0 a 2047). Las instrucciones correspondientes se llaman SPOKE,SDPOKE y SLPOKE. Especialmente en el modo supervisor se ruega tener cuidado, ya que las manipulaciones en direcciones protegidas pueden tener graves consecuencias. Las funciones PEEK trabajan siempre en el modo supervisor.

Ejemplos

```
LPOKE XBIOS(14,1)+6,0
```

--> Suprime el buffer del teclado (fija el indicador Head y Tail al comienzo del buffer.

O alternativamente:

```
REPEAT
UNTIL INKEY$=""
```

--> Suprime el buffer del teclado por selección de signos.

```
BYTE{x}
CARD{x}
```

```
INT{x}
LONG{x}
{x}
FLOAT{x}
SINGLE{x}
DOUBLE{x}
CHAR{x}

x: iexp
```

Con estos comandos se pueden leer determinados tipos de variables a partir de una dirección o se pueden asignar a una dirección. Como función (por ej. y=BYTE{x}) se lee de la memoria a partir de la dirección 'x'. Como comando (por ej. BYTE{x}=y) se escribe el valor 'y' en la memoria a partir de la dirección 'x'.

Cuando se emplea INT{}, CARD{}, LONG{}, {}, FLOAT{}, SINGLE{} y DOUBLE{} ha de prestarse atención de indicar solo direcciones pares, ya que sino se desencadena un error de dirección (tres bombas).

Tipo	Significado
BYTE{x}	Lee/escribe un byte.
CARD{x}	Lee/escribe enteros de dos bytes sin signo.
INT{x}	Lee/escribe enteros de dos bytes con signo.
LONG{x}	Lee/escribe valor entero de 4 bytes.
FLOAT{x}	Lee/escribe variables de punto flotante de 8 bytes en GFA-BASIC Formato 3.0.
SINGLE{x}	Lee/escribe variable de punto flotante de 8 bytes en formato IEEE-Single.
DOUBLE{x}	Lee/escribe una variable de punto flotante de 8 bytes en formato IEEE-Double.
CHAR{x}	Lee/escribe un string que termina con un byte cero. Tiene mucha importancia en la comunicación con rutinas C, GEM y GEMDOS.

Con x%=LONG{adr%} se le asigna a la variable x% la palabra larga que figura en la dirección adr%, y con LONG{adr%}=x% se escribe el valor de la variable x% como palabra larga en la dirección adr%.

Con las instrucciones SINGLE y DOUBLE es posible, leer o escribir un formato de números extraño. Algunos compiladores C conocen el formato IEEE-DOUBLE. Para convertir por ej. un número de GFA-BASIC en el formato SINGLE o DOUBLE y reportarlo así en forma hexadecimal, se puede escribir lo siguiente:

```
a$=SPACE$(4)
SINGLE{V:a$}=1.2345
PRINT HEX$(CVL(a$),8)
```

o

```
a$=SPACE$(8)
DOUBLE{V:a$}=1.2345
PRINT HEX$({V:a$},8)'
PRINT HEX$({V:a$+4},8)
```

Algunas de las funciones mencionadas anteriormente corresponden aproximadamente a las funciones PEEK, LONG{x} (o {x}) corresponde

aproximadamente a LPEEK(x) pero no trabaja en el modo supervisor. Por eso {x} es más rápido que LPEEK. Cuando se accede a rangos de memoria protegidos (0 a 2047) se desencadena un error de bus (dos bombas) y se visualiza así un error de programación.

Ejemplos

```
adr%=XBIOS(2)
t%=TIMER
FOR i%=1 TO 4000
  ~LPEEK(adr%)
NEXT i%
PRINT (TIMER-t%)/200
t%=TIMER
FOR i%=1 TO 4000
  ~{adr%}
NEXT i%
PRINT (TIMER-t%)/200
'
FLOAT{*x}=PI
```

--> La primera parte del ejemplo demuestra, que {} es más veloz que LPEEK, la segunda parte escribe indirectamente un número de punto flotante en una variable.

```
BYTE{XBIOS(2)+4160}=&HFF
CARD{XBIOS(2)+4320}=&HFFFF
LONG{XBIOS(2)+4480}=&HFFFFFFFF
'
a$="Prueba"+CHR$(0)
PRINT CHAR{V:a$};
b$=SPACE$(5)
CHAR{V:b$}="palabra"
PRINT b$,ASC(RIGHT$(b$))
```

--> Primero se imprimen directamente en la memoria de la pantalla algunos valores y aparecen allí como rayas. Entonces se asignan tanto a$ como también strings b$, que terminan con un byte cero, donde a$ se asigna convencionalmente (a través de +CHR$()) y se evidencia inconvencionalmente a través de CHAR{}, en cambio, b$ se asigna a través de CHAR{} y se evidencia con PRINT con el CHR$(0) visible.

```
VARPTR(x)      V:x
ARRPTR(y)       *y
```

x: nombre de una variable de cualquier tipo
y: nombre de una variable o de un arreglo con paréntesis vacío

Con ARRPTR(y) o con *y se determina la dirección de una variable 'y', en el caso de arreglos o strings se determina la dirección del descriptor.

Con VARPTR(x) o con V:x, en cambio, se determina en el caso de strings la dirección del string mismo o en el caso de arreglos la dirección de un elemento del arreglo.

En el caso de variables escalares (no campo o string) el significado de VARPTR, V:, ARRPTR y * es el mismo.

Ejemplo

```
DIM x%(10)
a$="Prueba"
PRINT ARRPTR(x%()),VARPTR(x%(0)),V:x%(1)
PRINT ARRPTR(a$),*a$,VARPTR(a$)
```

--> En la primera línea reporta la dirección del descriptor del arreglo así como la dirección de los dos primeros elementos del campo de x%(). En la segunda línea reporta dos veces la dirección del descriptor del string de a$ y una vez la dirección del primer byte de la cadena de caracteres.

ABSOLUTEx,y

x: una variable de cualquier tipo
y: iexp

Con la instrucción ABSOLUTE se cambia la dirección (ARRPR/*) de una variable.

Ejemplo

```
ABSOLUTE y,*x
x=13
y=7
PRINT x,y,*x,*y
```

--> Aquí se transfiere la variable 'y' a la dirección de 'x', de modo tal, que al final se reportan ambas variables con el mismo valor (7) y con la misma dirección.

Borrar e Intercambiar

CLEAR,CLR,ERASE
SWAP
SSORT,QSORT
INSERT,DELETE

CLEAR
CLRx[,y,...]
ERASEz1()[,z2(),...]

x,y: svar o avar
z1,z2: nombre de cualquier arreglo

Con CLEAR se borran todas las variables y los campos. Esta instrucción no se puede usar dentro de estructuras de repetición FOR-NEXT o de subprogramas. Al inicializar un programa se ejecuta automáticamente, sino su aplicación solo es razonable en el tratamiento de errores graves ,con RESUME x.

La instrucción CLR borra las variables, que se indican en la lista que le sucede. Pero con CLR no se pueden borrar arreglos. ERASE borra arreglos completos, los cuales se pueden volver a dimensionar después de este comando.

A diferencia de la versión 2.0 también se pueden borrar varios arreglos con la instrucción ERASE, por ej. ERASEx(),y().

Ejemplo

```
x=2
y=3
CLEAR
PRINT x,y
'
x=2
y=3
CLR x
PRINT x,y
'
DIM x(10)
PRINT FRE(0)
ERASE x()
PRINT FRE(0)
```

--> Imprime en pantalla tres veces el número 0 y una vez el número 3. Luego siguen dos números, que indican que el lugar de la memoria ocupado por DIM se volvió a liberar con ERASE.

```
SWAPa,b
SWAPe().f()
SWAP*c.d()
```

```
a,b  : avar o svar
c    : Indicador sobre un descriptor de arreglos
d,e,f: nombre de un arreglo
```

En su variante más sencilla la instrucción SWAP sirve para intercambiar dos variables del mismo tipo (SWAPa,b). Pero también se puede usar para intercambiar dos campos. El intercambio de campos es muy rápido, ya que solo se intercambian los respectivos descriptores. Así también se intercambia la dimensión de ambos arreglos. Los arreglos no deben estar dimensionados para esto.

C contiene el pointer sobre el descriptor del arreglo y d() es el nombre del arreglo en la tercera variante. Esta es interesante ante todo en la transferencia indirecta de campos a subprogramas.

La instrucción SWAP se debería diferenciar nítidamente de la función SWAP, que se ha tratado en el apartado sobre operaciones bit.

Ejemplos

```
x=1
y=2
PRINT x,y
SWAP x,y
PRINT x,y
```

--> Reporta después de 1 y 2 también los números 2 y 1.

```
DIM x(3)
change(*x())
PRINT x(2)
'
PROCEDURE change(adr%)
  SWAP *adr%,a()
```

```
  ARRAYFILL a(),1
  SWAP *adr%,a()
RETURN
```

--> El arreglo x() se completa con unos en el procedimiento change sin que aparezca el nombre x() en el subprograma. Así se puede emplear un subprograma para varios campos. Entonces se transfiere la dirección del descriptor, como en el ejemplo y mediante SWAP al comienzo y al final se accede a estos campos bajo un mismo nombre.

En la versión 3.0 se recomienda aquí la transferencia indirecta de campos como parámetro VAR.

```
DIM x(3)
CHANGE (x())
PRINT x(2)
'
PROCEDURE change (VAR a())
  ARRAYFILL a(),1
RETURN
```

```
QSORTa(s)[,n][,j%()]
QSORTx$(s)WITHi()[,n[,j%()]]

SSORTa(s)[,n][,j%()]
SSORTx$(s)WITHi()[,n[,j%()]]
```

```
a(): cualquier arreglo, también arreglo de strings
i(): arreglo de enteros ([,& o %)
j%(): arreglo de enteros de 4 bytes
x$(): arreglo de strings

n: iexp
s: +,- o ningún signo
```

Con las instrucciones SSORT y QSORT se pueden seleccionar los elementos de un arreglo según su tamaño. Con SSORT se emplea el procedimiento Shell-sort, con QSORT el procedimiento Quicksort.

En los paréntesis, delante de los cuales figura el nombre del arreglo a seleccionar, se puede agregar un signo más o menos. El signo menos indica que los elementos del arreglo se han de seleccionar en órden decreciente. En este caso aparece el elemento más grande después de la selección en el elemento cero del arreglo. El signo más indica, que el arreglo se debe seleccionar en órden creciente; en el elemento cero del arreglo figura entonces el valor más pequeño después de la selección. Si no aparece ningún signo se selecciona en órden creciente (como +).

El parámetro 'n' indica, que solo se deberán seleccionar los primeros elementos del arreglo (si vale OPTION BASE 0 éstos elementos son los que llevan los índices 0 a n-1, sino 1 a n).

Como tercer parámetro se puede indicar otro arreglo de enteros, que también se selecciona en la selección del primer arreglo. Cada intercambio de elementos en el primer arreglo también se lleva a cabo en el segundo arreglo.Esto se puede usar por ej. si en un arreglo aparece una clave de selección (código postal por ej.) y en varios aparecen

otras informaciones al respecto. El campo de enteros que se acopla en la selección contiene índices de arreglo, por ej., índices del fichero "R", valores LOC para SEEK etc.

En la selección de arreglos de strings se puede indicar un criterio de selección con WITH en forma de un arreglo con por lo menos 256 elementos. Sin la instrucción WITH se emplea la tabla ASCII normal como criterio de selección (ver segundo ejemplo).

Ejemplo

```
DIM x%(20)
PRINT "Sin selección:            ";
FOR i%=0 TO 10
  x%(i%)=RAND(9)+1
  PRINT x%(i%);"  ";
NEXT i%
PRINT
'
QSORT x%(),11
DIM index%(20)
PRINT "Selección descendente:  ";
FOR i%=0 TO 10
  PRINT x%(i%);"  ";
  index%(i%)=i%
NEXT i%
PRINT
'
SSORT x%(-),11,index%()
PRINT "Selección ascendente: ";
FOR i%=0 TO 10
  PRINT x%(i%);"  ";
NEXT i%
PRINT
PRINT "Campo acoplado:   ";
FOR i%=0 TO 10
  PRINT index%(i%);"  ";
NEXT i%
```

--> Evidencia un campo no seleccionado y dos hileras seleccionadas de números al azar. En una cuarta hilera figuran los valores de un campo que se acopló a la selección.

```
DIM b|(256)
FOR i%=0 TO 255
  b|(i%)=ASC(UPPER$(CHR$(i%)))
NEXT%
FOR i%=1 TO 7
  READ a$,b$
  b|(ASC(a$))=ASC(b$)
NEXT i%
DATA Ä,A,Ö,O,Ü,U,ä,A,ö,O,ü,U,,S
DIM n$(3)
FOR i%=0 TO 3
  READ n$(i%)
NEXT i%
DATA Muhlmann,Müller,Überlauf,Unterlauf
QSORT n$(),4
FOR i%=0 TO 3
```

```
  PRINT n$(i%),
NEXT i%
PRINT
QSORT n$() WITH b|(),4
FOR i%=0 TO 3
  PRINT n$(i%)
NEXT i%
```

--> Selecciona dos veces el arreglo de strings n$(), una vez sin WITH, una vez con. La indicación WITH lleva en este caso a la clasificación de las vocales alemanas con diéresis dentro de las correspondientes vocales sin diéresis y las minúsculas en las mayúsculas.

```
INSERTx(i)=y
DELETEx(i)
```

x: nombre de un arreglo
i: iexp
y: aexp o sexp, según el tipo de variable del arreglo

Con las instrucciones INSERT y DELETE se puede insertar o borrar un elemento en un arreglo. INSERT inserta en el arreglo 'x' el valor de la expresión y en la posición 'i'. En este caso se desplazan todos los elementos del arreglo, que tienen un índice mayor que 'i', una

posición más "arriba". Si un elemento figuraba antes en la posición i+3, por ej., entonces figura después de las instrucción INSERT en la posición i+4. El último elemento del arreglo se borra durante este procedimiento de inserción.

DELETE borra el elemento 'i' del arreglo 'x'. Todos los elementos del arreglo, que tienen un índice mayor que 'i', se desplazan una posición más "abajo". El último elemento del arreglo se iguala en este caso a cero (en campos de cadenas de caracteres se iguala a un string vacío). Estas dos instrucciones se adecúan especialmente para la administración de listas, en las cuales se deben insertar y borrar permanentemente elementos.

Ejemplos

```
DIM x%(5)
FOR i%=1 TO 5
  x%(i%)=i%
NEXT i%
INSERT x%(3)=33
FOR i%=0 TO 5
  PRINT x%(i%)
NEXT i%
```

--> Imprime en pantalla los números 0,1,2,33,3 y 4.

```
DIM x%(5)
FOR i%=1 TO 5
  x%(i%)=i%
NEXT i%
DELETE x%(3)
FOR i%=0 TO 5
  PRINT x%(i%)
NEXT i%
```

--> Imprime en pantalla los números 0,1,2,4,5 y 0.

Variables reservadas

```
FALSE,TRUE,PI
DATE$, TIME$,SETTIME
TIMER

FALSE
TRUE
TIMER
```

Las dos constantes FALSE y TRUE contienen valores para falso (0) y verdadero (-1) lógicos. La constante PI contiene el valor del número del círculo Pi.

Ejemplo

```
PRINT FALSE
IF TRUE
  PRINT PI
ENDIF
```

--> Imprime en pantalla los números 0 y 3.14159265359.

```
DATE$
TIME$
SETTIMEhora$,fecha$
DATE$=fecha$
TIME$=hora$
```

hora$,fecha$: sexp

La función DATE$ reporta la fecha del sistema en el formato:

TT.MM.JJJJ (día,mes,año) o MM/TT/JJJJ (formato americano, ver MODE)

TIME$ reporta la hora del sistema. Aquí está el formato:

HH:MM:SS (horas:minutos:segundos)

TIME$ va variando en intervalos de dos segundos.

Con la instrucción SETTIME se puede ajustar la hora y la fecha. En este caso, los strings hora$ y fecha$ deben tener el formato indicado recién en TIME$ y DATE$. Unicamente en las indicaciones del año entre 1980 y 2079 se puede suprimir la indicación del siglo. Si SETTIME se

usa con un formato incorrecto del string hora$ o fecha$, entonces no se modifican los valores actuales. Además también se puede modificar la fecha y la hora en forma individual a través de DATE$= y TIME=.

Ejemplo

```
PRINT DATE$,TIME$
SETTIME "20:15:30","27.2.1988"
PRINT DATE$,TIME$
```

--> Imprime en pantalla la fecha y la hora actuales del sistema, fija

de nuevo las variables y las vuelve a evidenciar.

TIMER

Con la función TIMER se obtiene la hora desde la conexión del sistema en dos centésimos de segundo.

Ejemplo

```
t%=TIMER
FOR i%=1 TO 2500
NEXT i%
PRINT (TIMER-t%)/200
```

--> Indica en segundos el tiempo que se nesecita para el loop vacío.

Casos especiales

LET, VOID, ~

Con LET también se pueden asignar valores en el caso de nombres de variables, que son iguales a palabras de instrucciones. Con VOID se puede usar una función, sin que se siga empleando su valor devuelto.

LETx=y

x: avar o svar

y: aexp o sexp

Con ayuda de LET se puede asignar a una variable el valor de una expresión. La expresión y la variable deben ser ambas numéricas o del tipo cadena de caracteres. Normalmente no es necesaria la instrucción LET. En dialectos anteriores de BASIC servía para poder emplear variables como palabras de instrucciones. Pero GFA-BASIC reconoce generalemente en forma automática si una palabra de la instrucción se debe emplear como variable.
Ejemplo

```
LET print=3
PRINT print
```

--> Imprime en pantalla el número 3.

```
VOIDfx
~fi
```

fx: aexp
fi: iexp

En lenguajes de programación se diferencian normalmente instrucciones de funciones. Las instrucciones llevan a la ejecución de cualquier actividad. En las funciones, ésta actividad consigue un valor , que se devuelve después. Este valor devuelto puede ser entonces elemento de una impresión, se puede evidenciar con PRINT o también se puede asignar a una variable con un signo igual, etc.

Pero en muchos casos no le interesa al programador el valor devuelto, sino solamente la actividad ejercida por la función. Así, por ej., la

función INP(2) reporta el código de una tecla pulsada. Si el programa debe esperar únicamente la pulsación de una tecla, entonces el código de la tecla que finaliza la espera no tiene importancia.En tal caso, se le puede indicar a GFA-BASIC a través de VOID que ejecuta la función pero que olvide el valor devuelto. En contraposición a VOID, cuando se emplea el tilde "~" se calcula un valor entero y luego se olvida. En VOID se realiza por ej. con INP(2) una conversión (innecesaria) a un número de punto flotante. El compilador reconoce por sí sólo estos cálculos innecesarios.

Ejemplo

```
VOID INP(2)
```

o mejor

```
~INP(2)
```

--> Solo espera a que se pulse la tecla sin la instrucción, innecesaria en este caso, dummy=INP(2). Con el tilde el intérprete se ahorra además la conversión del resultado en un valor de punto flotante.

BASDEPAGE
HIMEM

En la variable BASEPAGE figura la dirección del BASEPAGE del intérprete GFA-BASIC. El BASEPAGE es una zona de la memoria de 256 byte de longitud con la siguiente estructura:

Byte	Contenido
0 a 3	Dirección del comienzo del TPA (transient programm area)
4 a 7	Dirección del final del TPA más 1.
8 a 11	Dirección del segmento del TEXTO del programa.
12 a 15	Longitud del segmento del TEXTO.
16 a 19	Dirección del segmento DATA.
20 a 23	Longitud del segmento DATA.
24 a 27	Dirección del BSS (block storage segment).
28 a 31	Longitud del BSS.
32 a 35	Dirección del DTA (disk transfer adress).
36 a 39	Dirección del Basepage del proceso llamado.
40 a 43	Reservado.
44 a 47	Dirección del Environment-string.
48 a 127	Reservado.
128 a 255	Línea de comando (el primer byte indica la longitud del texto del comando).

En la variable HIMEM figura la dirección a partir de la cual GFA-BASIC no usa más la memoria. Este es un valor preseleccionado, que se encuentra 16384 bytes debajo de la memoria de la pantalla.

Ejemplo

```
a%=(BASEPAGE+&H2C)
DO
  a$=CHAR(a%)
  EXIT IF LEN(a$)=0
  PRINT a$
```

```
  ADD a%,SUCC(LEN(a$))      ! SUCC=+1
LOOP
```

--> Aquí se evidencia el environment completo.

Administración de memoria

```
FRE
BMOVE
BASEPAGE,HIMEM
RESERVE
INLINE
MALLOC,MSHRINK,MFREE
```

```
FRE()
FRE(x)
```

x: aexp

Esta función calcula el espacio libre de la memoria. El parámetro 'x' no se considera en este caso. Con FRE(x) se desencadena además una Garbage-Collection (recolección en el extremo superior de la memoria de las zonas de strings aún necesarias). FRE() indica el tamaño de la zona de memoria, que está libre sin esta garbage-collection.

Ejemplo

```
frei%=FRE(0)
max%=frei%/3/4
DIM x%(max%)
PRINT frei%,max%
```

--> Dimensiona un arreglo de tal manera, que ocupa aproximadamente un tercio del espacio libre de la memoria. Un campo de enteros de 4 byte ocupa 4 byte por elemento, por eso de divide por 4.

```
BMOVE ab_adr,an_adr,cant
```

ab_adr,an_adr,cant: iexp

Con ayuda de la instrucción BMOVE se pueden copiar zonas de la memoria. En la expresión ab_adr dice a partir de qué dirección figura la zona a copiar. En an_adr dice en qué dirección se debe dejar la copia de la zona de la memoria. En cant dice la cantidad de bytes a copiar.

La instrucción se ejecuta secuencialmente mucho más rápida con parámetros pares que con impares.También se puede usar cuando se superponen la zona original con la zona a copiar.

Ejemplo

```
DIM screen2%(64000/4)
adr%=VARPTR(screen2%(0))
FOR i%=0 TO 300 STEP 100
  PBOX 0,i%,639,i%+50
NEXT i%
BMOVE XBIOS(2),adr%,32000
BMOVE XBIOS(2),adr%+32000,32000
```

```
HIDEM
REPEAT
  IF MOUSEY<>my%
    BMOVE adr%+my%*80,XBIOS(2),32000
    my%=MOUSEY
  ENDIF
UNTIL MOUSEK=2
```

--> Cuando se mueve el ratón a lo largo del eje y, se hace el "scrolling por la pantalla" de una zona de la memoria.

RESERVE[n]

n: iexp

Con la instrucción RESERVE se puede indicar, cuán grande ha de ser la zona de memoria usada por GFA-BASIC. Cuando n es positivo, se reservan n bytes para el intérprete, el resto se deja libre. Cuando n es negativo, ésto tiene el mismo efecto que las dos instrucciones:

```
RESERVE
RESERVE FRE(0)-n
```

Cuando no se indica un parámetro se reestablece el mismo estado que al comienzo del intérprete.

La zona de la memoria se puede reservar solo en pasos de 256 bytes. La instrucción se puede usar, por.ej., para liberar una zona de la memoria para datos o resource-files.

Si se achicó la zona de la memoria para GFA-BASIC con RESERVE, se debería recordar de volver a agrandarlo luego, ya que sino el espacio disponible se hace cada vez más pequeño con cada llamada del programa.

Ejemplo

```
RESERVE 2560
EXEC 0,"\WORDPLUS.PRG","",""
RESERVE
```

--> Reserva 2560 bytes para el intérprete, carga WORDPLUS.PRG (si existe) e inicializa el programa. Cuando se interrumpe WORDPLUS.PRG se devuelve el lugar reservado de la memoria al programa llamado.

INLINEadr,anz

adr: variable entera de 4 bytes, no variable de campo
anz: constante entera menor que 32700

Detrás de esta instrucción no es posible hacer un comentario, ya que internamente se reserva exactamente en el lugar, en el que sino figuraría el comentario, el espacio de memoria deseado (o sea se completa con bytes cero). El espacio de memoria comienza siempre en una dirección par.

Al ejecutar INLINE se escribe esta dirección en la variable entera adr. Al almacenar o cargar el programa se acopla el espacio de memoria reservado.

Si se posiciona el cursor sobre la línea del programa que contiene la instrucción INLINE y se pulsa la tecla Help, entonces aparece en la línea superior del editor una línea del menú con las instrucciones LOAD,SAVE y CLEAR. Con los items del menú "LOAD" y "SAVE" es posible cargar o almacenar ficheros en la línea con la instrucción INLINE. En este caso se preseleccionó el sufijo de memoria reservado. En este espacio de la memoria se pueden cargar por ej. cuadros, tablas o programas assembler.

```
Malloc(x)
MFREE(y)
MSHRINK(y,z)
```

x,y,z: iexp

La función MALLOC (GEMDOS72) sirve para reservar (alocar) zonas de la memoria. Cuando el parámetro 'x' es igual -1, la función reporta la longitud del mayor espacio libre relacionado de la memoria. En este caso MALLOC es una función de consulta.

Cuando 'x' es un número positivo significa que se deben reservar 'x' bytes. En este caso reporta MALLOC la dirección inicial del espacio reservado de la memoria. Si aparece un error en el intento de reserva, se reporta cero.

Cuando se deben alocar espacios de memoria más grandes, se debe achicar primero el espacio de la memoria usado por GFA-BASIC. Espacios alocados de la memoria deberían ser liberados sin falta antes de la finalización del programa. Esto ocurre automáticamente cuando se abandona el intérprete.

MFREE (GEMDOS73) vuelve a desocupar el lugar reservado de la memoria. En este caso figura en 'y' la dirección inicial del bloque de memoria que se debe desocupar, o sea un valor devuelto obtenido con MALLOC. El valor devuelto es cero, cuando la desocupación se finalizó sin errores, o un número de error negativo.

MSHRINK (GEMDOS74) achica el espacio reservado de la memoria, que anteriormente había sido alocado con MALLOC. El parámetro 'y' contiene en este caso la dirección del espacio reservado de la memoria, que había sido devuelto en MALLOC. En 'z' figura la nueva longitud teórica del bloque de la memoria que se quiere achicar. La función MSHRINK reporta 0 cuando se ejecutó correctamente el achicamiento, -40 cuando se indicó una dirección errónea como 'y' y -67 cuando la longitud teórica nueva debe ser más grande que la longitud actual.

Es importante, que en MFREE y MSHRINK nunca se transfiera un pointer erróneo.

Ejemplo

```
RESERVE 1000
PRINT MALLOC(-1)
adr%=MALLOC(60000)
PRINT adr%
IF adr%>0
  x%=MSHRINK(adr%,30000)
  y%=MFREE(adr%)
ENDIF
```

RESERVE

--> Indica el tamaño de espacio de memoria libre relacionado más grande y luego la dirección de una zona reservada de la memoria de 60000 bytes, achica éste (si se reservó exitosamente) a 30000 y luego lo desocupa completamente.

3 - Operadores

Operadores son elementos de un lenguaje de programación que sirven para unir y comparar expresiones numéricas, lógicas y de cadenas de caracteres. En este apartado se presentan los operadores de GFA-BASIC en la siguiente secuencia:

En el primer apartado se tratan los operadores numéricos (+,-,*,/,^,DIV,\,MOD). Ellos unen cada vez dos expresiones numéricas y producen un número, que puede ser asignado por ej.con el signo igual de una variable. Los operadores + y - tienen una segunda función, se pueden emplear como signo.

El segundo apartado trata los operadores lógicos (AND,OR,XOR,NOT,IMP,EQV). Unen dos expresiones lógicas y producen un valor de verdad (verdadero o falso). En este caso ocupa un lugar especial el operador NOT. Solo actúa sobre una expresión lógica y niega su valor de verdad (de verdadero resulta falso y viceversa).

En el apartado 3 se analiza el operador de cadenas de caracteres. Este operador es el signo más (+), el cual une dos expresiones de cadenas de caracteres.

El siguiente apartado describe los operadores relacionales. Con su ayuda se pueden comparar dos expresiones numéricas o dos expresiones de cadenas de caracteres. Como resultado se obtiene un valor de verdad (verdadero o falso).

En el último apartado se trata la secuencia en la cual se ejecutan los operadores, cuando aparecen varios de ellos en una expresión numérica o lógica (jerarquía de los operadores).
Allí también se presentan los símbolos de los paréntesis "(" y ")" con los cuales se puede modificar la secuencia de la ejecución.

Los operadores aritméticos

```
+ - * / ^ DIV \ MOD
+ -
```

Los operadores aritméticos +, -, *, /, ^, DIV, \ y mod conectan dos expresiones numéricas y producen un número. Este número puede ser a su vez el componente de una expresión numérica o lógica, se puede asignar a una variable o se puede evidenciar, por ej., con PRINT. Los operadores + y - también se emplean como signos.

x+y	Produce la suma de los números x e y (adición).
x-y	Produce la diferencia x menos y (sustracción).
x*y	Produce el producto de x por y (multiplicación).
x/y	Produce el cociente de x dividido por y (división).
x^y	Produce la potencia de un número con base x e y como exponente (exponenciación).
x DIV y	Produce la parte entera del resultado de una división de
x\y	x dividido y (división entera).
x MOD y	Produce el resto de x dividido y (aritmética modular).

Para DIV y MOD valen las siguientes relaciones:

```
x DIV y = TRUNC(x/y)
x MOD y = x-y*TRUNC(x/y)
```

Ejemplo

```
13 DIV 4 devuelve el resultado 3
13 MOD 4 devuelve el resultado 1
```

+x Provee al valor x con un signo positivo, o sea da x.
-x Provee al valor x con un signo negativo. El número producido tiene entonces un signo negativo, si x era positivo y un signo positivo, si x era negativo.

Operadores lógicos

AND OR XOR NOT IMP EQV

Estos operadores lógicos trabajan a nivel de bit para valores enteros de 32 bit.

Los operadores conectan dos expresiones lógicas y producen un valor de verdad (verdadero o falso). Una excepción está dada por el operador NOT, que niega el valor de verdad de la expresión que figura detrás de él.

El valor numérico para lo "lógicamente falso" (FALSE) es 0. Cualquier otro valor numérico es interpretado como "lógicamente verdadero". La variable TRUE tiene en este caso el valor -1. Todos los operadores lógicos se pueden emplear en expresiones numéricas. Las operaciones lógicas se ejecutan en este caso bit a bit.

La acción de operadores lógicos se puede describir idóneamente en base a las llamadas tablas de verdad. En estas tablas figuran en las primeras columnas los valores de verdad de las expresiones conectadas (operandos) y en la última columna el valor lógico producido.

NOTx

x: iexp

El operador NOT (negación) niega la expresión lógica que le sucede. Es el único operador lógico, que tiene solo un argumento.

Modifica cada bit individual de su argumento.

```
x  NOT x

w   f
f   w
```

Ejemplos

```
PRINT NOT FALSE
PRINT NOT TRUE
```

```
PRINT NOT 0

--> En la pantalla se visualizan los números -1,0 y -1.

x=1
PRINT BIN$(x,2)
PRINT BIN$(NOT x,2)

--> 01
    10 se visualizan en la pantalla.

x%=17
PRINT BIN$(x%,8),x%
PRINT BIN$(NOT x%,8),NOT x%

--> Retorna los valores:
    00010001  17
    11101110 -18

xANDy

x,y: iexp

El operador lógico AND (conjunción) analiza, si dos expresiones
lógicas x e y son ambas verdaderas. Solo en este caso produce el
valor TRUE (lógicamente verdadero, -1). Si una de los dos o ambas
expresiones lógicas son falsas, también AND produce FALSE.

Con AND se compara cada uno de los 32 bits de sus argumentos.

x   y   xANDy

w   w     w
w   f     f
f   w     f
f   f     f

Ejemplos

PRINT TRUE AND -1
PRINT FALSE AND TRUE

--> En la pantalla se visualizan los números -1 y 0.

x=3
y=10
PRINT BIN$(x,4)
PRINT BIN$(y,4)
PRINT BIN$(x AND y,4),x AND y

--> Retorna los valores:
    0011
    1010
    0010

xORy

x,y: iexp
```

La instrucción OR (disyunción) analiza, si por lo menos una de dos expresiones lógicas x e y es verdadera. En este caso se produce el "verdadero lógico". Solo si ambos x e y son "lógicamente falsos", también x OR y produce el "falso lógico". A diferencia de XOR también se produce un "verdadero lógico" en el caso en que ambos x e y son verdaderos.

OR significa, que uno o ambos argumentos son verdaderos. También OR trabaja a nivel de bit.

```
x   y   x or y

w   w     w
w   f     w
f   w     w
f   f     f
```

Ejemplos

```
PRINT TRUE OR -1
PRINT FALSE OR TRUE
PRINT 0 OR FALSE
```

--> En la pantalla aparecen los números -1,-1 y 0.

```
x=3
y=10
PRINT BIN$(x,4)
PRINT BIN$(y,4)
PRINT BIN$(x OR y,4),x OR y
```

```
--> 0011
    1010
    1011 y el número 11 aparecen en la pantalla.
```

xXORy

x,y: iexp

Este operador analiza, si una de dos expresiones lógicas x e y es "lógicamente verdadera". En este caso x XOR y también da "lógicamente verdadero". Si x e y son ambos "lógicamente verdaderos" o ambos "lógicamente falsos", entonces XOR da "lógicamente falso".

La diferencia con OR consiste en que TRUE OR TRUE y TRUE XOR TRUE dan FALSE (XOR= disyunción exclusiva).

XOR da verdadero (solamente) cuando en uno de sus argumentos se fija el bit correspondiente. XOR también trabaja para cada uno de los 32 bits de sus argumentos en forma individual.

```
x   y   x XOR y

w   w     f
w   f     w
f   w     w
f   f     f
```

Ejemplos

```
PRINT FALSE XOR -1
PRINT -1 XOR 1
PRINT 0 XOR FALSE
```

--> En el monitor aparecen los números -1,0 y 0.

```
x=3
y=10
PRINT BIN$(x,4)
PRINT BIN$(y,4)
PRINT BIN$(x XOR y,4),x XOR y
```

--> Retorna los valores:

```
0011
1010
1001      9
```

x IMP y

x,y: iexp

El operador IMP (implicación) corresponde a una deducción lógica. Un deducción lógica de este tipo únicamente puede ser falsa cuando de una afirmación verdadera se deduce algo falso. Por lo tanto, x IMP y solo es falso cuando x es verdadero e y es falso.

IMP trabaja a nivel de bit. En contraposición a AND,OR,XOR y EQV aquí la secuencia es de importancia.

```
x   y   x IMP y

w   w       w
w   f       f
f   w       w
f   f       w
```

Ejwemplos

```
PRINT TRUE IMP -1
PRINT 0 IMP FALSE
PRINT TRUE IMP 0
```

--> En la pantalla aparecen los números -1,-1 y 0.

```
x=3
y=10
PRINT BIN$(x,4)
PRINT BIN$(y,4)
PRINT BIN$(x IMP y,4)
```

--> Retorna los valores:

```
0011
1010
1110
```

xEQVy

x,y: iexp

El operador EQV (equivalencia) dará como resultado "lógicamente verdadero", cuando las expresiones lógicas x e y tienen el mismo valor lógico.

EQV trabaja a nivel de bit y fija los bits que son iguales en ambos argumentos. Esto es exactamente lo opuesto a lo que ocurre con XOR, de tal manera que (x EQV y) tiene el mismo significado que (NOTxXORy).

A	B	A EQV B
w	w	w
w	f	f
f	w	f
f	f	w

Ejemplos

```
PRINT TRUE EQV FALSE
PRINT FALSE EQV FALSE
```

--> En la pantalla aparecen los números 0 y -1.

```
x=3
y=10
PRINT BIN$(x,4)
PRINT BIN$(y,4)
PRINT BIN$(x EQV y,4)
```

--> Retorna los valores:
0011
1010
0110

Operador de cadenas de caracteres

a$+b$

a$,b$: iexp

Con el operador + también se pueden conectar cadenas de caracteres. El resultado es una cadena de caracteres, que contiene a$ y b$.

Ejemplo

```
a$="GFA-"
PRINT a$+"BASIC"
```

--> En la pantalla aparece el texto 'GFA-BASIC'.

Operadores relacionales

= == >= <= <>

Con los operadores relacionales se pueden comparar expresiones numéricas, lógicas y de cadenas de caracteres entre sí. El resul-

tado de esta comparación es siempre un valor lógico (verdadero=-1 o falso=0). Una excepción la constituye el operador ==. Con él no se pueden comparar expresiones de strings.

```
x=y

x,y: exp
```

El operador = compara dos expresiones numéricas o de cadenas de caracteres en cuanto a igualdad. Cuando ambas expresiones son iguales, dará como resultado "lógicamente verdadero", caso contrario dará "lógicamente falso".

Ejemplo

```
x=6
IF 2=x/3
  PRINT "Ok"
ENDIF
PRINT 2=x/3
```

--> En la pantalla aparece Ok y -1.

```
x= =y

     x,y:  aexp
```

Compara dos expresiones numéricas en cuanto a una igualdad aproximada. Para esta comparación intervienen ocho y medio lugares después de la coma (28 bit de la mantisa del número de punto flotante). El empleo de = = es útil en aquellos casos en que se usan números de punto flotante, en los cuales pueden aparecer imprecisiones en el redondeo.

Ejemplos

```
PRINT 1.0000000001=1
PRINT 1.0000000001==1
```

--> En la pantalla aparecen los números 0 y -1.

```
a=SINQ(77)
b=SIN(RAD(77))
PRINT a=b
PRINT a==b
```

--> Se visualizan los números 0 (lógicamente falso) y -1 (lógicamente verdadero).

```
x<y
x>y
x<=y
x>=y

x,y:  exp
```

Estos operadores sirven para comparar la magnitud de expresiones numéricas y de cadenas de caracteres. En las expresiones numéricas se comparan los números, que se pueden calcular a par-

tir de ellas.

En el caso de las expresiones de strings la comparación se rige por el código ASC de los caracteres. El string "ABC"se trata como la secuencia de números 65,66 y 67. Si la afirmación dice "ABC">"AAA", entonces se comparan primero los dos primeros caracteres que tienen el código ASC 65. Luego se toma el siguiente carácter para la comparación. Ya que B (Código ASCII 66) tiene un código mayor que A, B se interpreta como "mayor", En este punto se interrumpe la comparación de los strings y el resultado de la afirmación es "lógicamente verdadero", por lo tanto, "ABC" es mayor que "AAA".

Un caso especial de la comparación de strings aparece cuando uno de los strings termina antes de que se descubran signos diferentes. Un ejemplo sería: "AA">"A". Esta afirmación es "lógicamente verdadera", ya que un signo no existente se considera el "signo menor". Hasta "A"+Chr$(0)>"A" es "lógicamente verdadero".

Con respecto a los operadores individuales:

x>y es verdadero, cuando x es mayor que y.
x<y es verdadero, cuando x es menor que y.
x>=y es verdadero, cuando x es mayor o igual que y.
x<=y es verdadero, cuando x es menor o igual que y.

Las formas de escribir x<=y y x=<y así como x>=y y x=>y tienen el mismo valor. También se puede convertir x><y en x<>y.

Ejemplo

```
PRINT "AAA">"aaa"
PRINT -1<=4-5
```

--> En la pantalla aparecen los números 0 y -1.

```
x<>y
```

x,y: exp

Este operador analiza, si dos expresiones numéricas o de strings son desiguales. Si es este el caso, la afirmación x<>y es "lógicamente verdadera". Si x e y son iguales, entonces x<>y dará "lógicamente falso". Las formas de escribir x<>y y x><y tienen el mismo valor.

Ejemplo

```
PRINT "Prueba"<>"prueba"
PRINT -1<>4-5
```

--> En la pantalla aparecen los números -1 y 0.

El operador de asignación

```
x=y
x: var
y: exp
```

El signo igual = no solo se puede usar como operador de comparación, sino también como operador de asignación. En este caso se determina el valor de la expresión y que figura a la derecha del signo "=", y este valor se asigna a la variable x que figura a la izquierda.

Expresiones numéricas se pueden asignar solo a variables numéricas, expresiones de cadenas de caracteres se pueden asignar solo a expresiones de strings.

Para la asignación también existe la instrucción LET, var=exp (comparar LET), que tiene el mismo valor y que también permite la asignación en variables como por ej. let o rem.

Ejemplo

```
x=LEN("Prueba")+1
a$="GF"+CHR$(65)
PRINT x,a$
```

--> En la pantalla aparece 7 y 'GFA'.

Jerarquía de los operadores

Cuando en una expresión se encuentran varios operadores se ejecutan en un órden determinado. Esta secuencia depende de la llamada jerarquía de los operadores. Los operadores, que tienen los primeros lugares dentro de esta jerarquía, se ejecutan en primer lugar.

La jerarquía es:

()	Paréntesis
+	Adición de cadenas de caracteres
+ < > => <= <>	Comparación de cadenas de caracteres
+ -	Signos
^	Exponenciación
* /	Multiplicación, división
DIV MOD	División entera y aritmética modular
+ -	Adición, sustracción
+ == < <= >= > <>	Operadores relacionales
AND OR XOR IMP EQV	Operadores lógicos
NOT	Negación

Con ayudas de los paréntesis es posible ejecutar en primer lugar aquellos operadores que tienen una jerarquía baja. En la secuencia también se reconoce la conocida regla "la multiplicación y división preceden a la adición y sustracción".

Ejemplo

```
PRINT 2+4*3
PRINT (2+4)*3
PRINT 2+(4*3)
PRINT 3*2^2
```

--> En la pantalla aparecen los números 14,18,14 y 12.

4 - Funciones numéricas

Funciones matemáticas

```
ABS,SGN
ODD,EVEN
INT,TRUNC,FIX,FRAC,ROUND
MAX,MIN
SQR
EXP,LOG,LOG(10)
SIN,COS,TAN,ASIN,ACOS,ATN,DEG,RAD
SINQ,COSQ
```

Hay funciones numéricas para las siguientes tareas: Las funciones numéricas ABS y SGN dan el valor absoluto y el signo de una expresión numérica. ODD y EVEN analizan, si un número es par o impar. INT,TRUNC,FIX y FRAC indican los lugares antes y detrás de la coma de expresiones numéricas. ROUND redondea una expresión.

MAX y MIN indican la mayor o la menor de varias expresiones y SQR calcula la raíz cuadrada de una expresión. Las funciones trigonométricas son ASIN,ACOS,SIN,COS,TAN y ATN. EXP y LOG calculan potencias y logaritmos del número de Euler. LOG10 calcula el logaritmo en base 10.

DEG y RAD sirven para convertir grados en radianes. RND,RANDOM,RAND y RANDOMIZE sirven para producir números al azar.

```
ABS(x)
SGN(x)
```

x: aexp

La función ABS da el valor absoluto de una expresión numérica. Esto lleva a los siguientes valores de retorno:

x es		luego ABS(x) es
negativo	-->	-x
igual cero	-->	0
positivo	-->	x

Con la función SGN se puede saber, cuál es el signo de una expresión numérica. Vale lo siguiente:

x es		luego SGN(x) es
negativo	-->	-1
igual cero	-->	0
positivo	-->	1

Ejemplo

```
x=-2
y=ABS(x)
PRINT SGN(x),ABS(5-3),SGN(ABS(x*3))  !ABS(x*3) ist gleich 6
```

--> En la pantalla se visualizan los números -1,2 y 1.

```
ODD(x)
EVEN(x)

x: aexp
```

Ambas funciones analizan, si la expresión numérica x es par o impar. ODD da el valor -1 (TRUE), cuando x es impar y 0 (FALSE), cuando x es par. EVEN da -1 para un x par y 0 para un x impar. El cero se trata como un número par.

x es		luego ODD(x) es	luego EVEN(x) es
par	-->	0 (FALSE)	-1 (TRUE)
impar	-->	-1 (TRUE)	0 (FALSE)
igual 0	-->	0 (FALSE)	-1 (TRUE)

Ejemplo

```
x=2
PRINT ODD(x),EVEN(-2),ODD(3*5), EVEN(-3*x)
```

--> En la pantalla se visualizan los números 0,-1,-1 y 0.

```
INT(x)
TRUNC(x)
FIX(x)
FRAC(x)

x: aexp
```

Estas funciones indican los lugares antes y detrás de la coma en expresiones numéricas. INT,TRUNC (y la función FIX idéntica a TRUNC) retornan un número entero. La función TRUNC corta los lugares de x después de la coma. INT retorna el mayor número entero, que es menor o igual que x. La diferencia entre TRUNC (FIX) y INT no se puede reconocer en valores positivos de x. En cambio, en un x negativo con lugares detrás de la coma aparece una diferencia. Así, TRUNC eliminaría en un x de -1,2 el lugar después de la coma y conservaría el valor -1. En cambio, INT busca el siguiente número entero menor que -1,2 y halla como resultado -2.

FRAC indica los lugares de x detrás de la coma, o sea elimina los lugares antes de la coma dejando la parte fraccionaria. El valor obtenido por FRAC tiene el mismo signo que x. FRAC es complementario de TRUNC y no de INT. Vale lo siguiente:

```
x = TRUNC(x) + FRAC(x)
```

Si se reeemplaza TRUNC por INT, no vale más la ecuación en el campo de los números negativos (por ej. para y igual -1,2).

Ejemplo

```
x=-1.4
y=TRUNC(1.3)
PRINT y,INT(x),FIX(3*x),FRAC(x-3)
```

--> En la pantalla aparecen los números 1,-2,-4 y -0,4.

```
ROUND(x[,n])

x:  aexp
n:  iexp
```

La función ROUND(x) da un valor redondeado. La variante ROUND(x,n) redondea la expresión 'x'con 'n' lugares detrás de la coma. Si 'n' es igual a cero, se redondea como en ROUND(x) a números enteros. Si 'n' es negativo, se redondea antes de la coma. Así, ROUND(155,-1) da el número 160 (comparar también CINT()=Redondeo con resultado entero).

Ejemplos

```
y=ROUND(-1.2)
PRINT y,ROUND(1.7)
```

--> Visualiza en la pantalla los valores -1 y 2

```
FOR i%=-5 TO 5
  PRINT i%,ROUND(PI*100,i%)
NEXT i%
```

--> Imprime en la pantalla la variable y la correspondiente expresión formateada. Además se multiplica el número Pi por 100 y se evidencia en forma redondea con i% lugares. Si i% es negativo, se redondea antes de la coma.

```
MIN(x[,y,z,...])
MIN(x$[,y$,z$,...])
MAX(x[,y,z...])
MAX(x$[,y$,z$,...])

x,y,z:  aexp
x$,y$,z$: sexp
```

La función MAX indica la mayor de las expresiones numéricas x,y,z,... , que se presentan en la lista de parámetros de esta función. MIN determina en forma análoga el menor valor. Las funciones MAX y MIN se pueden aplicar también en expresiones de cadenas de caracteres.

Ejemplo

```
x=3
y=MAX(3.5,5-4)
PRINT MIN(x,y),MAX(-1,x*2)
```

--> En la pantalla se visualizan los números 3 y 4.

```
SQR(x)

x:  aexp
```

Retorna la raíz cuadrada de la expresión numérica x. Si la expresión x es menor que cero, se visualiza un mensaje de error.

Ejemplos

```
x=9
y=SQR(x)
PRINT y,SQR(4*4)
```

--> En la pantalla aparecen los números 3 y 4.

```
PRINT SQR(SQR(16))
PRINT SQR(-2)
```

--> Se visualiza el número 2 y luego un mensaje de error.

```
EXP(x)
LOG(x)
LOG10(x)
```

x: aexp

EXP calcula la potencia x en base al número de Euler (e=2,1782818284...) y LOG produce la función inversa de ésta, o sea determina el logaritmo de x en base e (logaritmo natural). En forma análoga LOG10 retorna el logaritmo de x en base 10 (logaritmo decimal).

La expresión numérica x debe ser mayor que cero en las funciones logarítmicas, sino aparece un mensaje de error.

Ejemplo

```
x=2
y=EXP(2)
PRINT y,LOG10(2*5),LOG(x)
```

--> En la pantalla aparece 7,389056098931,1 y 0,6931471805599.

```
SIN(x)
COS(x)
TAN(x)

ATN(x)

RAD(grad)
SINQ(grad)
COSQ(grad)
```

x,grad: aexp

Estas son las funciones trigonométricas. La expresión numérica x se interpreta como radianes. Las funciones calculan:

```
SIN   --> seno
COS   --> coseno
TAN   --> tangente
ASIN  --> arcoseno
ACOS  --> arcocoseno
ATN   --> arcotangente
```

Para convertir un ángulo de radianes a grados se usa la función

DEG() o bien su función inversa RAD(). DEG(x) corresponde a (x*180/PI).

Las funciones SINQ y COSQ dan valores interpolados de seno o bien de coseno, para ello se emplea internamente del GFA-BASIC una tabla con los valores del seno en intervalos de grados. De acuerdo con el valor del argumento de la función GRAD se interpolan linealmente los valores intermedios en intervalos de 1/16 grados. Esta precisión es suficiente para la representación gráfica en la pantalla y no se diferencia de los valores calculados de SIN o COS , pero es notoriamente más veloz (aproximadamente 10 veces más veloz).

A diferencia de SIN y COS los parámetros de SINQ y COSQ figuran en radianes.

```
SINQ(grad)    equivale a   SIN(RAD(grad))
CISQ(grad)    equivale a   COS(RAD(grad))
```

Ejemplos

```
x=90
y=COS(x*PI/180)
z=270*PI/180
PRINT y,SIN(z),TAN(45),ATN(1/2)
```

--> Se visualizan los números 1,-1,1,619775190544 y 0.4636476090008.

```
alpha%=30
PRINT SINQ(alpha%)
```

--> En la pantalla aparece 0.5.

Generador de números al azar

```
RND[(x)]
RANDOM(x)
RAND(y)
RANDOMIZEy
```

```
x:  aexp
y:  iexp
```

Este grupo de comandos sirve para producir números al azar. RND produce un número al azar entre 0 y 1 (inclusive 0, exclusive 1). El parámetro opcional x no tiene significado.

RANDOM produce un número al azar entero entre 0 y x (inclusive 0, exclusive x). La expresión numérica x no debe ser entera, sino todos los números deben aparecer con la misma probabilidad.

RAND produce un número al azar entero de 16 bit entre 0 y y-1. Se evalúan 16 bit de y.

El comando RANDOMIZE inicializa el generador de números al azar con el valor 'y'. Así, cuando el generador de números al azar se inicializa varias veces con el mismo 'y', después siempre se obtiene la misma secuencia de números al azar. Al comienzo de cada

ejecución del programa se inicializa el generador de números al azar con un número elegido "al azar". Por lo tanto, si no se usa RANDOMIZE se obtienen después de cada inicialización del programa otros números al azar con RND,RANDOM o RAND.

Para la inicialización del generador de números al azar se puede usar RANDOMIZE también sin parámetros o RANDOMIZE0.

Ejemplos

```
x=RND
PRINT x,RND(2)
```

--> En el monitor aparecen dos números al azar entre 0 y 1.

```
x=RANDOM(2)
y=RAND(4)
PRINT x,y,RAND(x),RANDOM(3*x)
```

--> En la pantalla aparecen cuatro números al azar enteros.

```
RANDOMIZE 3
x=RND
RANDOMIZE 3
PRINT x,RND
```

--> Se visualiza dos veces el mismo "número al azar".

Aritmética de enteros

Comandos y funciones

```
DEC,INC
ADD,SUB,MUL,DIV
PRED(),SUCC()
ADD(),SUB(),MUL(),DIV(),MOD()
```

```
DEC,INC
ADD,SUB,MUL,DIV
```

Estos comandos equivalen a la forma abreviada de escribir las siguientes expresiones:

```
DEC x       equivale a    x=x-1
INC x       equivale a    x=x+1
ADD x,y     equivale a    x=x+y

SUB x,y     equivale a    x=x-y
MUL x,y     equivale a    x=x*y
DIV x,y     equivale a    x=x/y
```

Las expresiones que figuran a la izquierda requieren mucho más tiempo para su ejecución qwue las expresiones que figuran a la derecha. Esta diferencia es muy marcada en variables enteras.

Es importante saber que INC,DEC,ADD,SUB,MUL y DIV no realizan una prueba de desbordamiento en el caso de variables enteras (%,&,I).

DECi

```
INCi

i:  avar
```

Los comandos DEC e INC sirven para modificar un valor por uno.DEC disminuye el valor de la variable numérica i por uno, mientras que INC incrementa el valor de i por uno.

Estos comandos trabajan con variables de punto flotante, sin embargo son mucho más veloces con variables enteras.

Ejemplos

```
x%=4
y%=7
DEC x%
INC y%
PRINT x%,y%
```

--> En la pantalla aparecen los números 3 y 8.

```
a|=255
INC a|
INC a|
PRINT a|
```

--> Da 1!!, ya que no hay prueba de error.

```
ADDx,y
SUBx,y
MULx,y
DIVx,y

x:  avar
y:  aaxp
```

El comando ADD incrementa una variable x por el valor y, mientras que SUB disminuye a x por el valor y. MUL multiplica x con y y asigna el resultado a x. DIV divide la variable x por y, dejando el resultado en x.

En estos comandos x debe ser una variable numérica, mientras que y es una expresión numérica.

Estos comandos trabajan con variables de punto flotante, sin embargo son marcadamente más veloces con variables enteras.

Ejemplo

```
x%=1
y%=2
z%=3
ADD x%,y%          !x ahora es igual a 3
SUB z%,(x%-1)/2    !la expresión numérica (x%-1)/2 da 1
PRINT x%,y%
```

--> En la pantalla se visualizan los números 3 y 2.

```
PRED(),SUCC()
```

ADD(),SUB(),MUL(),DIV(),MOD()

PRED y SUCC dan el siguiente número más grande y el siguiente número más pequeño. ADD(),SUB(),MUL(),DIV() y MOD() ofrecen una rápida aritmética de enteros con notación polaca.

Estas dos expreciones pueden ser usadas con expresines string (vease apartado sobre administración de cadenas de caracteres)

Ejemplo

```
i%=6
j%=PRED(i%)
PRINT j%,SUCC(2),PRED(3*i%)
```

--> En la pantalla aparecen los valores 5, 3 y 17.

```
ADD(x,y)
SUB(x,y)
MUL(x,y)
DIV(x,y)
MOD(x,y)
```

x,y: iexp

Estas funciones se pueden emplear en lugar de algunos operadores numéricos:

```
ADD(x,y)   equivale a   x+y
SUB(x,y)   equivale a   x-y
MUL(x,y)   equivale a   x*y
DIV(x,y)   equivale a   x\y       x DIV y
MOD(x,y)   equivale a   x MOD y
```

Ya que las funciones mencionadas anteriormente trabajan con aritmética de enteros, se ignoran los lugares detrás de la coma. De acuerdo con las instrucciones

```
x%=5
y%=4
ADDy%,3
z%=SUB(x%,3)
```

x% tiene el valor 5, y% tiene el valor 7 y z% tiene el valor 2.

Las funciones ADD,SUB,MUL,DIV y MOD se pueden encasillar o agrupar de cualquier forma. Esta secuencia de operadores y operandos se llama notación polaca.

Ejemplos

```
DEFINT "a-z"
x=4
y=ADD(x,x)    !y wird gleich 8
z=SUB(x,2)
PRINT y,z,ADD(x,MUL(y,2))
```

--> En la pantalla aparecen los números 8,2 y 20.

```
DEFINT "a-z"
x=2
y=MUL(x,3)                    !y es igual a 6
PRINT y,DIV(8,x),MOD(11,4)    !MOD(11,4) es igual a 3
```

--> En la pantalla aparecen los números 6,4 y 3.

```
DEFINT "a-z"
x=5
 y=ADD(SUB(x,2),MUL(3,4))
PRINT y,DIV(8,MOD(14,4))
```

--> En la pantalla aparecen los números 15 y 4.

Operaciones de bits

```
BCLR,BSET,BTST,BCHG
SHL,SHR,ROL,ROR
AND(),OR(),XOR(),IMP(),EQV()
SWAP()
BYTE(),CARD(),WORD()
```

Las operaciones de bits influyen sobre expresiones numéricas a nivel de bit. Los comandos (conocidos para los programadores de Assembler) BCLR,BSET,BTST y BCHG borran,fijan,analizan y niegan bits. SHL,SHR,ROL y ROR desplazan o rotan bits.

Las funciones AND,OR,XOR, IMP y EQV son conexiones lógicas. En este caso vale la siguiente forma de contar: 0 es el bit de menor valor; en variables enteras de 4 byte es 31 el bit de mayor valor y al mismo tiempo el bit del signo (si se fijó el bit del signo, se representa el número negativo en el complemento de dos, sino es un número positivo).

SWAP intercambia las dos palabras de un valor de 4 byte. BYTE lee los 8 bits inferiores y CARD los 16 bits inferiores de una expresión entera. WORD amplia una palabra a palabra larga, o sea el bit 15 se copia en bit 16 hasta bit 31.

```
BCLR(x,y)
BSET(x,y)
BCHG(x,y)
BTST(x,y)

x,y:  iexp
```

Estos comandos permiten borrar, fijar, negar y analizar un bit. En este caso el recuento de los bits comienza en cero. El número del bit se halla dentro del rango 0 a 31 y se enmascara internamente en el procesador con AND31.

El bit no. y de la expresión numérica x se iguala a cero con la función BCLR. En forma análoga, el número de bit y de x se iguala a uno con BSET.Con BCHG el bit y de x se iguala a uno, si anteriormente era cero o se iguala a cero, si anteriormente era uno. La función BTST da -1(TRUE), cuando el bit y de x es igual a 1 y 0(FALSE), cuando este bit es igual a cero.

Ejemplos

```
x=BSET(0,3)
PRINT x,BSET(0,5)
```

--> En el monitor se visualizan los números 8(2^3) y 32(2^5).

```
REPEAT
  t|=Inp(2)
  PRINT CHR$(t|),CHR$(BCLR(t|,5))
UNTIL CHR$(t|)="x"
```

--> Pulsando una tecla de letras aparece la letra y la correspondiente mayúscula (en minúsculas siempre se fijan 5 bit y el borrado de este bit exige la conversión en una mayúscula).

```
s$="Palabra"
FOR i%=1 TO LEN(s$)
  PRINT CHR$(BCHG(ASC(MID$(s$,i%)),5));
NEXT i%
```

--> En la pantalla se visualiza "pALABRA", o sea convierte cada minúscula en mayúscula y viceversa. Sin embargo, esto no vale para diéresis.

```
SHL(x,y)   SHL&(x,y)   SHLI(x,y)
SHR(x,y)   SHR(x,y)    SHRI(x,y)
ROL(x,y)   ROL&(x,y)   ROLI(x,y)
ROR(x,y)   ROR&(x,y)   RORI(x,y)
*
x,y:  iexp
```

Estas instrucciones desplazan(Shift) o rotan(Rotate) por y bit el contenido de una expresión numérica x . Cuando no se indica un tipo de variable especial ocurre ésto en la longitud de palabra larga (4 bytes), cuando se indica '&' ocurre en longitud de palabra (2 bytes) y cuando se indica 'i' en la longitud de un byte.La tercera letra del nombre de la función indica la dirección del desplazamiento o de la rotación. En este caso 'L' significa izquierda (left) y 'R' significa derecha (right).

En las funciones Word (&) se copia a continuación el bit 15 en los bits 16 a 31, y en las funciones de byte (I) se borran los bits 8 a 31.

Los siguientes ejemplos numéricos muestran el efecto de los comandos Shift.

```
x%    SHLI(x,1)   BIN$(x%,16)          BIN$(SHLI(x%,1),16)
18    36          00000000 00010010    00000000 00100100
642   4           00000010 10000010    00000000 00000100

x%    SHL&(x%,1)  BIN$(x%,16)          BIN$(SHL&(x%,1),16)
18    36          00000000 00010010    00000000 00100100
130   4           00000000 10000010    00000001 00000100

x%    SHR&(x%,2)  BIN(x%,16)           BIN$(SHR&(x%,2),16)
24    6           00000000 00011000    00000000 00000110
```

```
4162  1040        00010000 01000010     00000100 00010000
```

(Los bits se agruparon en grupos de ocho solamente para que se comprenda con mayor facilidad. BIN$ no agrupa).

Los siguientes ejemplos numéricos se relacionan con los comandos de rotación. Aquí los bits que abandonan la correspondiente zona de números se vuelven a introducir en el lado opuesto.

Se fija por ej. solo el mayor bit de un byte. Este byte se rota entonces hacia la izquierda por un bit (ROLI(128,1)). El bit desplazado por la izquierda se vuelve a introducir por la derecha, de tal manera que queda fijado el primer bit del resultado de la función. Con SHLI(128,1) el bit fijado que se desplazó hacia afuera entonces sería igual a cero.

Otros ejemplos numéricos serían:

```
xI   ROLI(x,1)    BIN$(xI,8)     BIN$(ROLI(xI,1),8)
6    12           00000110       00001100
130  5            10000010       00000101

xI   RORI(xI,3)   BIN$(xI,8)     BIN$(RORI(xI,3),8)
66   144          01000010       00000000
2    64           00000010       01000000
```

Ejemplo

```
x|=128+1                        !fija Bit 7 y 0
x|=ROR|(x|,1)                   !y es igual a 192
PRINT SHL(y%,4),y%*2^4
PRINT SHL(ROR|(128+1,1),4)
```

--> En el monitor aparece tres veces el número 3072. La expresión a*2^b equivale a la función SHL(a,b), siempre que no se desplace ningún bit sobre la zona de validez de 4 bytes. La formulación, que usa la función de bit es marcadamente más veloz.

```
AND(x,y)
OR(x,y)
XOR(x,y)
IMP(x,y)
EQV(x,y)
```

x,y: iexp

Estas funciones son conexiones lógicas de dos expresiones numéricas. En el resultado de AND solo se fijan bits, que se han fijado tanto en x como en y. El resultado de la función de OR contiene bits en los lugares, en los cuales se fijaron bits en x o en y o en ambas expresiones. XOR solo fija bits, que se han fijado en x o en y pero no en x e y simultáneamente. Dicho con otras palabras: XOR fija bits, que tienen distintos valores en x y en y.

IMP solo asigna cero a bits, cuando se fijó el correspondiente bit en x y no en y, en los demás casos se fija el bit en el resultado de la función. EQV fija un bit en el resultado de la

función, cuando los correspondientes bits en x e y tienen los mismos valores. Para la ilustración de este comando observe también las tablas de valores de verdad en el apartado sobre operadores lógicos.

Ejemplos

```
x=3
y=2
z=AND(x,y)  !z es igual a 2
PRINT OR(2,7),XOR(x,1+4+8)
```

--> En la pantalla se visualizan los números 7 y 14.

```
PRINT BIN$(15,4),15
PRINT BIN$(6,4),6
PRINT BIN$(IMP(15,6),4),"IMP(15,6)"
PRINT BIN$(EQV(15,6),4),"EQV(15,6)"
```

--> En la pantalla aparece:

```
    1100    15
    0101    6
    0111    IMP(15,6)
    0110    EQV(15,6)
```

SWAP(x)

x: iexp

La función SWAP interpreta la expresión numérica x como palabra larga (4 bytes) e intercambia su palabra superior e inferior (cada vez 2 byte). Esta función no tiene nada que ver con el comando del GFA-BASIC del mismo nombre y se usa para diferentes fines(por ej. para transferir a una rutina del sistema de trabajo un parámetro de palabra larga en forma de dos palabras o para tratar los pointer con una secuencia de palabras intercambiada).

Ejemplo

```
x=1044480
PRINT BIN$(x,32)
y=SWAP(x)
PRINT BIN$(y,32)
```

--> En el monitor aparece:

```
00000000000011111111000000000000
11110000000000000000000000001111
```

Un típico ejemplo para sus uso es:

```
~WIND_SET(0,13,SWAP(t%),t%,0,0)
```

BYTE(x)
CARD(x)
WORD(x)

BYTE retorna los 8 bits inferiores de la expresión numérica x. CARD lee en forma equivalente los 16 bits inferiores de x. WORD amplía una palabra a la longitud de palabra larga (o sea el bit 15 se copia en bit 16 a 31).

Ejemplo

```
PRINT BYTE(1+254),BYTE(1+255)
PRINT HEX$(CARD(&H1234ABCD))
```

--> En la pantalla se imprime 255,0 y ABCD.

5 Administración de las cadenas de caracteres

LEFT$,RIGHT$
MID$(como función)
PRED,SUCC
LEN
INSTR,RINSTR
STRING$,SPACE$,SPC
UPPER$
LSET,RSET,MID$(como comando)

Los comandos LEFT$ y RIGHT$ retornan la porción izquierda o derecha de una cadena de caracteres. Si MID$ se usa como función, puede retornar porciones que aparecen en el medio de un string. Si MID$ se usa como comando , se puede insertar con él un string dentro de otro string. PRED y SUCC retornan el carácter con el siguiente valor de ASCII menor o respectivamente mayor que él de la expresión de string indicado.

LEN determina la longitud de una cadena de caracteres, INSTR y RINSTR buscan la primer ocurrencia de un string dentro de otro string y responderá con la posición donde se halla la coincidencia. STRING$,SPACE$ y SPC sirven para producir cadenas de caracteres, que contienen repetidamente la misma éxpresión string'. UPPER convierte todos los símbolos de un string en mayúsculas. La inserción de un string a la izquierda y a la derecha de otro se lleva a cabo con LSET y RSET.

LEFT$(a$[,x])
RIGHT$(a$[,x])

a$; sexp
x: iexp

LEFT$ devuelve los primeros caracteres de x de la cadena de caracteres a$. Si x es mayor que la cantidad de caracteres en a$, entonces se retorna todo a$. Sin la indicación de x, sólo se reporta el primer carácter del string a$. En forma análoga, RIGHT$ reporta solo los últimos caracteres de a$. Sin la indicación de x se determina el último carácter de a$.

Ejemplos

```
a$="Manual para GFA-BASIC"
b$=LEFT$("GFA-Systemtechnik",4)
PRINT b$;RIGHT$(a$,5)
```

--> En la pantalla se imprime la palabra 'GFA-BASIC'.

```
a$="Oberhausen"
b$=LEFT$(a$)+RIGHT$("Technik")
PRINT b$
```

--> En la pantalla aparece ok'.

MID$(a$,x[,y])(como función)

a$: sexp
x,y: iexp

La función MID$ devuelve el carácter 'y' a partir de la posición x del string a$. Cuando x es mayor que la longitud de a$, se producirá una cadena de caracteres nula. Si se suprime y, entonces se retorna toda la porción del string a partir del carácter y.

Ejemplo

```
a$="Manual para GFA-BASIC"
b$=MID$(a$,13,9)+MID$("Versión 3.0",8)
PRINT b$
```

--> En la pantalla aparece el texto 'GFA-BASIC3.0'.

```
PRED(a$)
SUCC(a$)
```

a$: sexp

PRED devuelve el carácter del valor de ASCII disminuido en uno de la 'expresión string' a transferir.En este caso solo se evalúa el primer carácter de una cadena de caracteres. En forma análoga, SUCC devuelve el valor aumentado en uno. En este caso PRED(a$) equivale a la expresión CHR$(PRED(ASC(a$))) y SUCC(a$) a CHR$(SUCC(ASC(a$))).

De estas dos funciones también se dispone para expresiones enteras(ver apartado de aritmética de enteros).

Ejemplo

```
caracter$="B"
anterior$=PRED(caracter$)
posterior$=SUCC(caracter$)
PRINT anterior$,caracter$,posterior$
```

--> En la pantalla aparecen las letras A,B y C.

```
LEN(a$)
TRIM$(a$)
```

a$: sexp

LEN calcula, cuántos caracteres están contenidos en a$ y retorna el número.

TRIM$ elimina espacios en blanco en el márgen izquierdo y derecho del string.

Ejemplos

```
a$="Prueba"
x=LEN(a$)+1
PRINT x,LEN("Wort")
```

--> En la pantalla aparecen los números 7 y 6.

```
b$="   prueba   "
PRINT LEN(b$)
PRINT TRIM$(b$)
PRINT LEN(TRIM$(b$))
```

--> Se visualiza 12, el string 'Prueba' sin espacios en blanco y 6.

```
INSTR(a$,b$)
INSTR(a$,b$,[x])
INSTR([x],a$,b$)

a$,b$:  sexp
x:  iexp
```

La función INSTR busca la primer ocurrencia del string a$ dentro del string b$, de tal manera que se inicia la búsqueda en a$ a partir del carácter x, sino la búsqueda comienza en el primer carácter. Responderá con la posición, a partir de la cual b$ está contenido en a$. Si no se halló b$, INSTR es igual a cero. Si a$ y b$ son strings nulos, INSTR es igual a uno.

Ejemplo

```
a$="GFA-Systemtechnik"
x=INSTR(a$,"System")
PRINT x,INSTR("GFA-BASIC","BASIC",6)
```

--> En la pantalla aparecen los números 5 y 0.

```
RINSTR(a$,b$)
RINSTR(a$,b$,[x])
RINSTR([x],a$,b$)

a$,b$:  sexp
x:  iexp
```

RINSTR() lleva a cabo una búsqueda igual que INSTR, pero comenzando al final de la cadena de caracteres.

Ejemplo

```
PRINT RINSTR("A:\ORDNER\*.GFA","\")
```

--> Busca la última barra invertida'\' en el nombre Pfad y retorna la posición del carácter buscado(en este caso 10).

```
STRING$(x,a$)
STRING$(x,code)
SPACE$(x)
SPC(x)

x,code:  iexp
a$:  sexp
```

La función STRING$ produce un string, que contiene la expresión string 'a$' o el código del valor de ASCII, reproducido x veces (entre 0 y 32767). Con la indicación 'code' se aplica CHR$(code).

SPACE$ produce un string que contiene x espacios en blanco. SPC produce en un comando PRINT x espacios en blanco. A diferencia de SPACE$, SPC no puede producir un string, que se pudiera asignar a una variable por medio de un signo igual.

Ejemplo

```
a$="GFA "
b$=SPACE$(5)
PRINT b$;STRING$(3,a$);SPC(4);STRING$(5,"<")
```

--> En la pantalla aparece ' GFA GFA GFA <<<<<'.

```
UPPER(a$)
```

a$: sexp

Convierte todas letras minúsculas de una cadena de caracteres en letras mayúsculas. Esto también funciona con diéresis.

Ejemplo

```
a$="Palabra"
b$=UPPER$(a$)+UPPER$(" de"+" prueba")
PRINT b$;UPPER$(" Gfa-Basic 3.0")
```

--> En la pantalla se visualiza
'PALABRA DE PRUEBA GFA-BASIC3.0'.

```
LSETa$=b$
RSETa$=b$
MID$(a$,x[,y]) (como comando)
```

a$: svar
b$: sexp
x,y: iexp

LSET y RSET insertan la expresión string b$ hacia la izquierda (LSET) o bien hacia la derecha (RSET) de la variable de strings a$ y completa eventualmente con espacios en blanco.

MID$ empleado como comando posibilita la inserción de una expresión string en el medio de una variable de strings. Así en el caso de

```
MID$(a$,x,y)=b$
```

se insertaría la expresión del string b$ a partir del carácter x en la variable a$. El parámetro opcional y fija cuántos caracteres de b$ se han de insertar como máximo en a$. Si se suprime y, se transfieren a a$ todos los caracteres posibles. Con MID$ no se modifica la longitud de a$, sino que los caracteres provenientes de b$ transcriben partes de a$. Cuando a$ es demasiado corto como para absorber todo el b$, entonces se interrumpe la inserción.

Ejemplos

```
a$="      "
FOR i%=1 TO 128         ! columnas izquierdas
  LSET a$=STR$(i%)
  PRINT a$;
NEXT i%
PRINT
FOR i%=1 TO 128
  RSET a$=STR$(i%)      ! columnas derechas
```

```
  PRINT a$;              ! de números
NEXT i%
~INP(2)
```

--> Retorna columnas de números formateadas a la izquierda y a la derecha. En lugar de RSETa$=STR$(i%) también se puede usar a$=STR$(i%,5,0).

```
a$="GF   ASIC"
MID$(a$,3)="A-B"
b$="Palabra "
MID$(b$,7,2)="onda"
PRINT a$,b$
```

--> Se visualizan las palabras 'GFA-BASIC' y 'Palabron'.

6 - Ingreso de datos a través del teclado y salida de datos a través de la pantalla

En este capítulo se presentan las posibilidades básicas de ingreso y de salida de datos. En primer lugar se explica el rol de INKEY$, que sirve para comprobar si se activó alguna tecla del teclado. Luego se solicita una variable mediante INPUT y las instrucciones emparentadas LINE INPUT, FORM INPUT y FORM INPUT AS.

El análisis de las posibilidades de salida de datos comienza con la instrucción más simple, con PRINT. A continuación se tratan sus versiones ampliadas (PRINT AT, PRINT USING). Luego se habla de las instrucciones que solicitan (CRSCOL,CRSLIN,POS) o determinan la posición del cursor (TAB,HTAB,VTAB).

Al final de este apartado se presentan los comandos KEYxxx. Con ayuda de estas nuevas instrucciones es posible una manipulación sencilla de las funciones dependientes del teclado durante la ejecución del programa.

INKEY$

INKEY$ lee un carácter del teclado. Sin embargo esta función no tiene la capacidad de comprobar si se activaron las teclas de conmutación del teclado(Shift,Alternate,Control,CapsLock). INKEY$ no espera que se pulse una tecla sino que reporta un string nulo,si no se pulsó ninguna tecla desde la última vez que se quizo comprobar que había sido activado el teclado. De lo contrario la función reporta el símbolo de ASCII de la tecla presionada. Si la tecla pulsada no tiene código ASCII, como por ej. las teclas de función o las teclas Help o Undo, se obtiene el código Scan de la tecla pulsada. En este caso se retorna un string de dos caracteres de longitud, que contiene el código CHR$(0) en el primer carácter y la correspondiente caracterización de las tecla especial en el segundo carácter. El siguiente ejemplo ilustra cómo se obtienen valores con INKEY$.

Ejemplo

```
DO
  t$=INKEY$
  IF t$<>""
    IF LEN(t$)=1
      PRINT "ASCII-Code: ";ASC(t$),"Codigo ASCII: ";t$
    ELSE
      PRINT "CHR$(0), Scan-Code: ";CVI(t$)
    ENDIF
  ENDIF
LOOP
```

--> En cada tecla pulsada indica su código ASCII o Scan. Nota: CVI(t$) equivale aquí a ASC(RIGHT$(t$)), ya que ASC(t$)=0.

```
INPUT["text",]x[,y,...]
INPUT["text';]x[,y,...]
```

x,y: avar o svar

La instrucción INPUT se puede emplear en varias variaciones. Sirve para ingresar variables o listas de variables con o sin explicaciones

preliminares("text"). INPUT siempre se refiere a la última posición del cursor. Con ayuda de PRINT AT, seguido de un semicolon o LOCATE,VTAB,HTAB se puede ubicar el cursor para fijar la posición de ingreso.

Cuando a la palabra INPUT le sucede un texto, se lo puede separar de las siguientes variables por una coma o un semicolon. Cuando se emplea un semicolon, se agrega a la descripción un signo de interrogación y un espacio en blanco y el cursor se posiciona detrás de ello. Cuando se emplea una coma, el cursor se posiciona directamente detrás del último carácter de la descripción.

Cuando no figura ningún texto detrás de INPUT, aparece en cada caso un signo de interrogación, seguido de un espacio en blanco, detrás del cual está el cursor.

Si solo se solicita una única variable, se debe confirmar su ingreso pulsando las teclas Return o Enter. Cuando se solicitan varias variables con una instrucción INPUT, entonces se puede confirmar cada variable pulsando Return o Enter, pero las variables también se pueden separar con una coma y se pueden ingresar pulsando una única vez la tecla Return o Enter. Si se quieren ingresar comas en un string mediante la instrucción INPUT, se debe emplear el comando LINE INPUT.

Si se esperaba una variable numérica y se ingresó un texto, suena una señal de error, y el ingreso se debe iniciar nuevamente. Hasta la pulsación de la tecla Return o Enter se pueden borrar caracteres dentro del texto a ingresar con las teclas Backspace o Delete; las teclas izquierda y derecha del cursor también son activas. Pulsando la tecla Insert se puede conmutar entre el modo Inserción y normal. La longitud de ingreso máxima vale 255 caracteres.

Caracteres especiales se pueden ingresar de tres formas diferentes:

- Pulsando <Alternate> y los números del bloque numérico, por ej.: Presionando la tecla Alternate pulsar el número 64 en el bloque de decenas. Cuando se deja de pulsar la tecla Alternate aparecerá @. Esto también vale para INKEY$, INP(2), diálogos GEM, etc. en la medida en que no se desconecta con KEYPAD.

- Ingresando <Control><S> y otro símbolo, por ej.:<Control><S><c> para el símbolo Pi.

- Ingresando <Control><A> y el siguiente código ASCII del carácter deseado, dependiendo de KEYPAD, comparar allí por ej.: <Control><A><2><7> para el símbolo Escape.

Ejemplo

```
INPUT a$
INPUT "",b$
INPUT "Dos números por favor: ";x,y
PRINT a$,b$,x,y
```

--> Lee dos strings y dos variables numéricas. El primer comando INPUT produce un '?', el segundo no produce texto y el tercero pide 'Dos números por favor:?'.

LINE INPUT["text",]a$[,b$...]
LINE INPUT["text";]a$[,b$...]

a$,b$: svar

LINE INPUT es una variante de la instrucción INPUT. A diferencia de INPUT, en este caso es posible ingresar comas en una variable string. Las descripciones previas, el ingreso de variables o de una lista de variables, la correción del ingreso hasta que se pulsa la tecla Return o Enter y otras posibilidades de variación se manejan igual que en INPUT, aunque solo es posible con variables string.

Ejemplo

```
LINE INPUT a$
INPUT b$
PRINT a$,b$
```

--> Por favor, ingrese dos veces el texto 'Kom,ma'. Luego aparece 'Kom,ma'y 'Kom' en el monitor, comparar también con LINE INPUT#.

FORM INPUTn,a$
FORM INPUT,ASa$

n: iexp
a$: svar

FORM INPUT y FORM INPUT AS sirven ambos para ingresar variables string. En este caso indica n, cuántos caracteres puede tener como máximo el string a$ (entre 1 y 255).

FORM INPUT AS retorna además el valor actual de a$, que se puede editar entonces. Las posibilidades de edición de ambas instrucciones son las mismas que las de la instrucción INPUT.

Ejemplo

```
FORM INPUT 10,a$
b$="Prueba"
FORM INPUT 5 AS b$
PRINT a$,b$
```

--> Solicita dos strings. Cuando se solicita el segundo string se visualiza la palabra 'Prueba' como valor a editar de b$.

PRINT
PRINT expresión
PRINT AT(columna,línea);expresión
WRITEexpresión
LOCATElínea,columna

expresión: cualquier sexp o aexp o combinación de ellos
columna,línea: iexp

La instrucción PRINT sin otros parámetros hace que se avance a la siguiente línea. Si el cursor ya se encuentra en la última línea, toda la pantalla se desplaza una línea más arriba. PRINT seguido de una

expresión produce la salida de esta expresión en la posición actual del cursor. Las cadenas de caracteres se deben encerrar en comillas. Cuando la expresión que debe salir está compuesta de varios elementos (constantes,variables o expresiones), se pueden separar las partes individuales con semicolon, coma o apóstrofe.

Cuando se emplea la coma se ubica el cursor detrás de la siguiente posición de columna divisible por 16. Cuando se alcanza la última columna, se mueve el cursor a la columna 17 de la siguiente línea. Cuando se emplea el semicolon se produce la salida de los correspondientes elementos sin espacio en blanco. Cuando se usa el apóstrofe, se inserta un espacio en blanco entre los correspondientes elementos.

PRINT AT hace posible posicionar la expresión que debe salir a partir de una columna y línea determinadas. Según la precisión se disponen en este caso de hasta 80 columnas y 25 líneas. El empleo de ventanas también restringe la zona de valores para la ubicación del cursor.

Cuando la expresión de salida no va seguida de un semicolon, se ubica el cursor al comienzo de la línea siguiente. Si ya se encontraba en la última línea, la pantalla se desplaza una línea hacia arriba.
Cuando se indican caracteres de control (códigos hasta 31) junto con PRINT, éstos se procesan a través del VT-52-Emulator (ver anexo).

Contrariamente con lo que ocurre con PRINT AT, LOCATE solamente ubica el cursor en la posición de línea y columna indicada. La salida de expresiones no es posible con LOCATE (comparar VTAB/HTAB).

La instrucción WRITE sirve para almacenar datos en archivos secuenciales para la lectura con INPUT. Después de la instrucción WRITE figuran expresiones numéricas y string, separadas una de la otra con comas.

En la salida se separan las expresiones entre sí con comas; expresiones string se encierran en comillas. Nota: Este formato es especialmente adecuado en la salida sobre disco para volver a leer con INPUT. Detrás de la última expresión de una instrucción WRITE también puede figurar un semicolon, entonces se reprime la salida final de CR/LF.

Ejemplos

```
a$="GFA-Systemtechnik"
PRINT Left$(a$,4)+"BASIC"'1+2;
PRINT ".0","GFA-";UPPER$(MID$(a$,5))
```

--> En la pantalla se visualiza el texto 'GFA-BASIC3.0' y
'GFA- SYSTEMTECHNIK'.

```
PRINT AT(8,4);"Texto en la cuarta columna, octava linea"
```

--> Escibe un string en la posición 4,8.

```
LOCATE 8,4
PRINT "en la octava linea, cuarta columna"
```

--> Posiciona el cursor en la cuarta columna de la octava línea y con el siguiente PRINT se visualiza un string en esta posición.

```
WRITE 1+1,"Que tal ?",3*4
```

--> Produce la impresión de 2,"Hallo",12.

PRINT USINGformato$,expresión[;]
PRINT AT(línea,columna);USINGformato$,expresión[;]

formato$: sexp
expresión: cualquier cantidad de sexp o sexp separadas con comas.
columna,línea: iexp

PRINT USING y su variación PRINT AT USING sirven para la salida formateada de datos en el monitor. Estas instrucciones trabajan básicamente como PRINT o PRINT AT. Sin embargo, los datos de la expresión que debe salir se formatean de acuerdo con el contenido de formato$.

Para formatear expresiones numéricas existen los siguientes símbolos:

Espacio para un número. Si es el último número de la instrucción de formato, entonces se redondea en la salida.

caracteres de #.

, Inserta una coma en el correspondiente lugar entre los caracteres de # y produce así una separación en los dígitos de las milésimas.

- El signo menos sólo debe figurar en el primer o último lugar del string de formato. Reserva un lugar para la salida de un signo negativo. En cambio, con un signo positivo sale un espacio en blanco.

+ Análogamente al signo menos, en números positivos se agrega antes o después el signo más. No se pueden combinar signos más y menos.

* Reemplazo de #, cero no significativos se reemplazan por asteriscos, en lugar de espacios en blanco.

$ Produce la salida de un carácter de $ antes de indicar el número, si figura inmediatamente antes del primer símbolo #.

^ Sirve para conectar el formato exponencial (E+000). Los caracteres de # determinan en este caso la longitud de la mantisa. Los símbolos ^ indican la cantidad de dígitos para el exponente , inclusive para E+ o E-. Si figuran varios # antes del punto fijo, se adapta el exponente de tal manera, que sea divisible por esta cantidad. Para números negativos debe preverse indefectiblemente un signo.

Contrariamente a la versión 2.0 ahora vale el principio: Formateo antes de exactitud. Quiere decir, que cuando hay desbordamiento solo salen los números en la correspondiente zona.

Para el formateo de cadenas de caracteres existen los siguientes símbolos:

& Produce la impresión de toda la cadena de caracteres.

! Restringe la impresión al primer carácter del string.

\...\ Indica cuántos caracteres de un string (inclusive los dos caracteres \) se imprimirán.

_ El subrayado produce la impresión del siguiente carácter de la indicación de formato.

Además es posible la impresión inalterada de inserciones de texto entre estas indicaciones de formato. Las variables, listas de variables o expresiones que le suceden al string de formato se separan con comas.

Ejemplo

```
PRINT USING "#.####",PI
PRINT AT(4,4); USING "PI_._._. #.###",PI;

--> Se visualizan los textos '3.1416' y 'PI...3.142'.

FOR i%=1 TO 14
  PRINT USING "###.##^^^^ ",2^i%;
NEXT i%

--> Salida:
          1.00E+00   2.00E+00   4.00E+00   8.00E+00   16.00E+00
         32.00E+00  64.00E+00 128.00E+00 256.00E+00  512.00E+00
          1.02E+03   2.05E+03   4.10E+03   8.19E+03   16.38E+03
```

Con MODE se pueden intercambiar comas y puntos.

MODEn

n: iexp

Con MODE se puede elegir entre la notación de punto fijo y de punto flotante y entre coma decimal y punto en milésimos. Con MODE se selecciona además el formato de la representación de la fecha. Punto y coma valen para PRINT USING y STR$(x,v,n). El ajuste de la fecha vale para DATE$,SETTIME,DATE$= y FILES.

El parámetro n puede asumir valores entre 0 y 3. Para ello resultan los ajustes según la siguiente tabla:

Parámetro n	USING	DATE$
MODE 0	#.###.##	16.05.1988
MODE 1	#,###.##	05/16/1988
MODE 2	#.###,##	16.05.1988
MODE 3	#.###,##	05/16/1988

DEFNUMn

n: iexp

DEFNUM influencia la salida de números a través de la instrucción PRINT y sus variantes. Todas las salidas de números, que le suceden a una instrucción DEFNUM, se ejecutan con n dígitos (lugares antes y detrás de la coma, el punto no se cuenta). La precisión interna de

cálculo no se modifica por eso. Se accede al n+1-simo lugar por redondeo.

Ejemplo

```
PRINT 100/3
DEFNUM 5
PRINT 100/3
```

--> En la pantalla aparecen los números 33.333333333 y 33.333.

CRSCOL
CRSLIN
POS(x)
TAB(n)
HTABcolumna
VTABlínea

n,columna,línea: iexp
x: aexp

El grupo de instrucciones CRSCOL, CRSLIN, POS y TAB sirve para solicitar y ubicar el cursor. CSRCOL indica la posición de columna actual y CSRLIN la posición de línea actual del cursor. POS da el número de caracteres de salida en el monitor desde el último retorno de carro (carriage Return) AND255. La expresión x se ignora.

Sin embargo, el valor devuelto por POS no debe coincidir con la posición actual del cursor. Si se imprime por ej. una cadena de caracteres con 120 caracteres, el cursor figura en la columna 40, pero POS da 120. Especialmente en la salida de caracteres de control del cursor POS(0) tiene poco que ver con la columna en que se encuentra el cursor, ya que solo se cuentan los caracteres. En este caso no se presta atención a un LF(CHR$(10)), un CR (CHR$(13))retorna el contador a cero y un BS(CHR$(8)) disminuye el contador por uno.
TAB(n) evidencia tantos espacios en blanco, hasta que POS(0) alcanzó el valor de n. Si se superó n, primero se avanza una línea (CR/LF) igual que con PRINT sin parámetro. Nuevamente se emplea solo AND255.

Las instrucciones HTAB y VTAB posicionan el cursor en una determinada posición de columna o línea.

Cuando se posiciona el cursor el extremo superior izquierdo tiene la coordenada 1.

Ejemplos

```
PRINT AT(38,12);"Prueba ";
PRINT CRSCOL'CRSLIN
PRINT Tab(37);"Prueba ";
PRINT POS(0)
```

--> En el medio de la pantalla aparece 'Prueba 45 12' y 'Prueba 44'.

```
PRINT AT(4,3);"Palabra 1"
HTAB 4
VTAB 2
PRINT "Palabra 2"
```

--> Evidencia el texto 'Palabra 1' en la cuarta columna de la tercera línea y el string 'Palabra 2' en la cuarta columna de la segunda línea.

Las instrucciones KEYxxx

Este grupo de instrucciones hace posible el control a través del estatus de las teclas de conmutación del teclado durante la ejecución del programa y la instalación de las teclas de función con cadenas de caracteres libremente definidas (como máximo 31 caracteres por vez), que también están a disposición en el editor del GFA-BASIC.

KEYPADn

n: iexp

La expresión numérica n se evalúa bit a bit y tiene el siguiente significado:

BIT	Significado	0	1
0	NUMLOCK	sí	no
1	NUMLOCK	no conmutable	conmutable
2	CTRL-KEYPAD	normal	cursor
3	ALT-KEYPAD	normal	ingreso ASCII
4	KEYDEF sin ALT	no	sí
5	KEYDEF con ALT	no	sí

Cuando se fija bit 1 está conectado el modo NUMLOCK, o sea el bloque de números está instalado como en una 'PC'(ver también el apartado sobre el editor).

Cuando se fija bit 2, se puede manejar el cursor pulsando Control y una tecla con una flecha.

Cuando se fija bit 3 se pueden ingresar caracteres ASCII pulsando las tecla Alternate seguido del valor de ASCII.

Cuando se fija bit 4 se pueden ocupar con strings tanto F1 a F10 como Shift-F10 a Shift-1. Esto vale con bit 5 únicamente cuando se pulsa además la tecla Alternate.

El ATARI ST tiene normalmente la conexión que equivale a KEYPAD 0. Mediante el intérprete del GFA-BASIC se preselecciona en el editor el KEYPAD 46 (o sea se fijan los bits 1,2,3 y5).

KEYTESTn
KEYGETn
KEYLOOKn

n: ivar

La función KEYTEST equivale a INKEY$ y lee un carácter del teclado, si se pulsa una tecla(menos Alternate,Shift y Caps-Lock) desde el último ingreso. Se reporta cero, si no se pulsa ninguna tecla, caso contrario se reporta el valor ASCII del carácter (como KEYGET).

KEYGET espera a que se pulse una tecla. En n se retorna una palabra de doble densidad con la siguiente estructura: Bit 0-7:Código ASCII,Bit8-

15:cero,Bit16-23:Código Scan,Bit24-31:Estatus de las teclas de conmutación del teclado(kbshift).

KEYLOOK posibilita la lectura desde el buffer del teclado sin que se modifique o borre su contenido (retorno igual que en KEYGET).

Cuando se emplea KEYTEST,KEYGET y KEYLOOK y se indica una variable de bytes o de palabra automáticamente ocurre una conversión en un valor entero de 2 bytes (como WORD(BIOS(2,2)).

Ejemplos

```
PRINT "Oprima <Esc>"
REPEAT
UNTIL INKEY$=CHR$(27)
'
PRINT "Oprima otra vez <Esc>"
'
REPEAT
  KEYTEST n|
UNTIL n|=27
```

--> Espera dos veces la pulsación de la tecla.

```
PRINT "Oprima una tecla"
tecla_1|=INP(2)
PRINT "Oprima otra tecla"
KEYGET tecla_2|
PRINT "INP(2): ";tecla_1|
PRINT "KEYGET: ";tecla_2|
```

--> Espera dos veces la pulsación de una tecla y reporta los códigos de las teclas oprimidas.

```
REPEAT
  ' calcular
  KEYLOCK n%
UNTIL n%
LINE INPUT "Que quiere ?",a$
```

--> Espera en un loop de cálculo simulado una pulsación de la tecla y lee luego un string sin suprimir la primera tecla.

```
DO
  KEYGET a%
  PRINT HEX$(a%,8)'BIN$(a%,32)'
  OUT 5,a%
  PRINT
LOOP
```

--> Espera la pulsación de una tecla,reporta el correspondiente código en hexadecimal y binario e indica el carácter. Nota: A través de OUT 5,igual que con VID, se indican caracteres de control del cursor (<32).

KEYPRESSn

n: iexp

La instrucción KEYPRESS simula la pulsación de una tecla. La expresión numérica n contiene además el valor ASCII de la tecla, que se simula oprimir. En Highword se pueden transferir adicionalmente el código Scan y el estatus de las teclas de conmutación del teclado (comparar KEYGET).

Ejemplo

```
FOR i&=65 TO 90      ! simula la pulsacion de
  KEYPRESS i&        ! las letras grandes
NEXT i&              ! A hasta Z
'
KEYPRESS 13          ! simula la pulsacion de
'                    ! la tecla Return
LINE INPUT a$        ! lee caracteres hasta el retorno de linea
'                    ! por medio de chr$(13) o RETURN
PRINT a$

KEYPRESS &H3B0000    ! Oprime F1
KEYPRESS &1001B      ! Oprime la tecla Esc, permitiendo la
                     ! lectura de teclas de GEM



KEYDEF 1,"Que tal ?"+CHR$(13)
KEYPRESS &H83B0000
PAUSE 1
LINE INPUT a$
PRINT a$
```

--> Define un texto, que se debe visualizar al pulsar Alternate-F1, luego simula la pulsación de esta tecla, espera finalmente que la correspondiente rutina de interrupción pueda ejecutar la definición de las teclas y lee luego "Que tal ?" con INPUT.

KEYDEFn,s$

n: iexp
s$: sexp

La instrucción KEYDEF hace posible la instalación de las teclas de función con cadenas de caracteres cualesquiera, cuya longitud no supere los 31 caracteres. Detrás de la instrucción KEYDEF se indica además una expresión numérica entre 1 y 20 inclusive. Con n de 1 a 10 se puede acceder a las teclas de función F1 a F10. Con n de 11 a 20 se pueden indicar valores para la tecla Shift y para las teclas de funciones F1 a F10.

Uno dispone de la instalación de las teclas de función durante la ejecución del programa y en el editor. Sin embargo, en el editor del GFA-BASIC sólo si se pulsa adicionalmente la tecla Alternate (comparar KEYPAD Bit 4 y 5)

Ejemplo

```
KEYDEF 1,"F1"
KEYDEF 11,"Shift+F1"
```

--> Pulsando la tecla de función F1 se visualiza el texto 'F1' y pulsando la tecla Shift y F1 (cada vez en combinación con Alternate) se visualiza el texto 'Shift+F1'.

Ingreso general y salida general de datos

En el siguiente capítulo se analiza primero la lectura de constantes (DATA,READ,RESTORE). A continuación, se trata la administración de índices (DIR$,CHDIR,DIR,FILES,MKDIR,RMDIR). En relación con estas instrucciones se explica la estructura del sistema jerárquico de archivos.

Luego se analizan la apertura, el cierre y el renombramiento de archivos (EXIST,KILL,NAME,OPEN,CLOSE). Además se indican las posibilidades de ingreso y de salida de zonas de la memoria (BLOAD,BSAVE,BGET,BPUT).

A continuación se presentan las posibilidades de acceso secuenciales (INPUT$,INPUT#,PRINT#) y secuenciales de índice (SEEK,RELSEEK) así como archivos de random-access (FIELD,GET#,PUT#,SEEK#,RELSEEK#). El apartado sobre periféricos trata las posibilidades de entrada y salida byte a byte como INP,OUT o sus funciones de pregunta INP?,OUT? y la recepción de los caracteres a través de la interfase seriada y MIDI (INPAUX$,INPMID$).

Finalmente se explica la administración del ratón y del joystick (MOUSE,MOUSEX,MOUSEY,MOUSEK,HIDEM,SHOWM,STICK,STRIG) así como la salida en una impresora (LPRINT,LPOS,HARDCOPY).

Líneas de datos

```
DATAconst[,const1,const2,...]
READvar[,var1,var2,...]
RESTORE[mar]
```

const,const1,const2: constantes numéricas o alfanuméricas (strings)

var,var1,var2: avar o svar

marca: nombre definido por el usuario

Con DATA se pueden almacenar valores constantes a la vez que se ahorra espacio. La lectura de estos datos ocurre con READ. Se pueden presentar valores numéricos en forma hexadecimal, octal o binaria. Si se debe leer una cadena de caracteres, que contiene comas, toda la cadena de caracteres debe figurar entre comillas.

A las instrucciones DATA y READ les pertenece un indicador de datos. Este siempre señala al siguiente valor de DATA que debe leer READ. Al comenzar el programa éste es el primer valor detrás del primer DATA.

Con ayuda de la instrucción RESTORE se puede dirigir el indicador de datos a determinadas líneas del DATA. Con este fin se fija una marca antes de esta línea del DATA y el indicador de datos se fija con

RESTOREmar. Si después de una instrucción RESTORE no figura ninguna marca, el indicador de datos se dirige al primer valor de DATA que aparece en el programa.

La secuencia de caracteres mar puede estar compuesta de números, letras, subrayados y puntos y, a diferencia de los nombres de variables, también puede comenzar con un número. La marca misma debe finalizar con 'dos puntos'.

Ejemplos

```
FOR i=1 TO 3
  READ a
  PRINT a'
NEXT i
'
RESTORE letras_romanas
READ a$,b$,c$,d$
PRINT
PRINT a$'b$'c$'d$
'
DATA 1,2,3,4
DATA a,b,c,d
'
letras_romanas:
DATA I,II,III,IV
```

--> En un loop se leen los valores numéricos 1,2 y 3 y se indican en la pantalla. Luego se dirige el indicador de datos mediante RESTORE a la línea del DATA con los números romanos. Esto ocurre con números_romanos. Estos datos se asignan a continuación a las variables strings a$,b$,c$ y d$ y se visualizan.

```
DATA 10,&A,$A,&HA,&012,&X1010,%1010
FOR i%=1 to 7
  READ a%
  PRINT a%
NEXT i%
```

--> Lee siete veces el número 10. En READ/INPUT/VAL etc. se pueden caracterizar números HEX con $ y números BIN con % y no solo con &H o & o bien &X.

Administración de archivos

A continuación se explicarán las instrucciones que sirven para organizar archivos. Para ello se debe conocer la estructura de especificaciones de archivos según las reglas de sistema jerárquico de archivos. Una especificación de archivos consiste de tres partes: la indicación de la disquetera (disk-drive), el nombre del archivo y la caracterización del archivo. La indicación de la disquetera contiene la caracterización de la disquetera A a O seguida de dos puntos. El nombre del archivo tiene 1 a 8 caracteres. Opcionalmente se puede indicar una caracterización del archivo. Consiste de un punto seguido de 1 a 3 caracteres.

Para seguir subdividiendo archivos existen índice (también llamados ordenadores). Estos ordenadores se pueden instalar a distintos niveles; el nivel más bajo también se llama índice básico. A partir de

este índice básico se puede producir la ramificación en otros subíndices. Por lo tanto, una inscripción de un índice puede estar compuesta de

- Indicaciones de la disquetera
- Nombres de subíndices
- Nombre y caracterización del archivo

Estas partes se separan con barras invertidas "\"(Backslash). Los nombres para índices tienen el mismo formato que los nombres de archivos. Así se obtiene el camino de acceso a un archivo por acoplamiento de los elementos presentados anteriormente. Primero la indicación de la disquetera, seguido del nombre para subíndices, y luego el nombre del archivo, por.ej:

A:\TEXTOS.DOC\MANUAL\CAPITULO_1.DOC

En este ejemplo el camino de acceso es:

A:	Indicación de la disquetera
\TEXTOS.DOC	Indice TEXTOS.DOC
\MANUAL	Subíndice MANUAL
\CAPITULO_1	Nombre del archivo CAPITULO_1

Se dispone además de dos símbolos especiales, que hacen posible una selección de archivos más fácil. Pueden figurar dentro de nombres de archivos y sus caracterizaciones. Son el signo de interrogación y el asterisco (el signo de multiplicación).

El signo de interrogación hace posible que cualquier carácter , cuyo código ASCII es mayor que 32, ocupe esta posición. También es posible usar el signo de interrogación en varios lugares de la especificación del archivo dejar que se sucedan unos a otros varias veces seguidas. El asterisco significa que cualquier cadena de caracteres puede ocupar todas las posiciones restantes.

Indices

DFREE(n)
CHDRIVEn o n$
DIR(n)
CHDIRnombre$

n: iexp
nombre$: sexp

DFREE (disk free) da el lugar de almacenamineto libre de una estación de disco en bytes. Esta función puede tardar cierto tiempo en devolver un resultado, especialmente en particiones de harddisk.

CHDRIVE (change drive) fija la disquetera estándar. Si en una instrucción de entrada o salida no se indica ningúna disquetera especial, el ingreso y la salida se ejecuta en la disquetera estándar. Con CHDRIVE también se puede indicar un string, cuyo primer carácter es la letra inicial de la disquetera como número de disquetera.

DIR$(n) consigue el camino de acceso actual para una disquetera, como fue fijada por CHDIR.

Para DFREE(n) y DIR$(n) y CHDRIVEn el valor n es la caracterización de la disquetera. En este caso n puede asumir los valores entre 0 y 16, donde 0 equivale a la disquetera activa y 1 a 16 equivalen a las disqueteras A a P.

CHDIR fija el índice actual. Ya que con CHDIR no es posible ningún cambio de disquetera, esta instrucción siempre vale para la disquetera actual o para la disquetera indicada. Así CHDIR "B:\TEST" cambia el ordenador por defecto para la disquetera B:, o sea el ordenador, al que se acude cuando se accede a esta disquetera sin indicaciones del camino, o sea sin \. En este caso la expresión string nombre$ cumple la misma función que el camino de acceso deseado. Cuando nombre$ solo contiene una barra invertida "\", se conmuta al índice básico de la disquetera actual.

Además hay dos nombres especiales para ordenadores "." y "..". El nombre de ordenador "." representa una forma de designación alternativa para el ordenador activo y el nombre de ordenador ".." para el ordenador superior.

Si en un disco existe por ej. el ordenador \TEST\01 y \TEST\02 y el ordenador \TEST\01 está activo (mediante CHDIR), entonces se puede cambiar al ordenador \TEST\02 con CHDIR"..\02".

Ejemplos

```
CHDRIVE 1
PRINT DFREE(0)
PRINT DIR$(2)
CHDRIVE "C:\"
```

--> CHDRIVE1 fija como disquetera estándar a la disquetera A.PRINT DFREE(0) indica el espacio libre de almacenamiento (en bytes) de la disquetera estándar (n=0. Ahora debido a CHDRIVE1 es la disquetera A:) en la pantalla. PRINT DIR$(2) da el camino de acceso en la disquetera B:.A continuación se fija la disquetera C: como disquetera estándar.

```
CHDIR "\"
CHDIR "TEXTOS.DOC\MANUAL"
CHDIR "ANEXO"
```

--> CHDIR "\" pasa al índice básico de la disquetera actual. En la segunda línea se pasa al subordenador del índice TEXTOS.DOC. En la tercera línea se sigue ramificando en el subíndice ANEXO(\TEXTOS.DOC\MANUAL\ANEXO).

DIRp$[TOnombre$]
FILESp$[TOnombre$]

p$,nombre$: sexp

Las instrucciones DIR y FILES sirven para la salida de índices. Cuando se usa DIR se visualiza en el pantalla en forma estándar el índice de una determinada disquetera. En este caso se indica el camino de acceso deseado en la expresión string p$. Lo mismo vale para la instrucción

FILES, aunque en este caso se indican adicionalmente la fecha, la hora y la longitud de los correspondientes archivos. Además se visualizan los nombres de ordenadores precedidos por un *, cuando cumplen con la máscara de búsqueda (por.ej.*.*), y eventualmente también los ordenadores "." y "..". Cuando la máscara del archivo p$ termina en ":" o en "\",entonces el mismo GFA-BASIC agrega "*.*" (comparar FSFIRST y FSNEXT).

Para DIR y FILES opcionalmente también es posible agregar TO nombre$. Cuando se aplica este agregado es posible derivar el índice a un archivo o a un aparato periférico. La expresión string nombre$ contiene en este caso el nombre de un archivo o bien de una caracterización de periféricos.

Ejemplo

```
DIR "A:\TEXTVER\*.DOC"
DIR "A:\TEXTVER\HANDBUCH\*.DOC" TO "B:\HANDBUCH\INHALT.ASC"
DIR "A:\*.*" TO "PRT:"
FILES "A:\*.DOC" TO "LST:"
```

--> En la primera línea se visualizan todos los nombres del archivo con la caracterización ".DOC"(del ordenador "\TEXTVER"en la disquetera A) en la pantalla. La segunda línea produce la derivación del índice a un archivo "INHALT.ASC" en la disquetera B. La realización en una impresora ocurre en una tercera línea. En la cuarta línea se derivan todos los nombres del archivo con la caracterización ".DOC" a la impresora.

FGETDTA()
FSETDTA(adr)

adr: iexp

La función FGETDTA() da la DTA (disk transfer adress).

Con FSETDTA se fija la DTA. También un Fileselect-box modifica la DTA. Para DTA se preseleccionó BASEPAGE+128. Esta dirección también la usa DIR,FILES y EXIST.

La DTA tiene la siguiente estructura:

Bytes	Offset	Significado
21	0	Reservado para GEMDOS
1	21	Atributos del archivo (ver más adelante)
2	22	Hora
2	24	Fecha
4	26	Longitud del archivo
14	30	Nombre del archivo finalizando con byte nulo, sin espacios en blanco.

El significado de los bits del atributo es:

Bit	Significado
0	Archivo que no se puede escribir, está protegido.
1	Archivo escondido.

2 Archivo de sistema.
3 Nombre del disco.
4 Ordenador.
5 Bit del archivo.

FSFIRST(p$,attr)
FSNEXT()

p$: sexp
attr: iexp

La función FSFIRST hace posible buscar el primer archivo, que satisface el criterio de búsqueda indicado en p$ (por ej. C:*.GFA). Nombres de archivos encontrados y otras informaciones se escriben en la DTA. El parámetro attr contiene los atributos, que pueden tener los archivos a encontrar.

La función FSNEXT() busca el próximo archivo, que satisface las condiciones de FSFIRST.

Ejemplo

```
~FSETDTA(BASEPAGE+128)                  ! setzt DTA
e%=FSFIRST("\*.GFA",-1)                 ! Suchkriterium festlegen
DO UNTIL e%
  IF BYTE{BASEPAGE+149} AND 16          ! wenn es ein Ordner ist
    PRINT "*";CHAR{BASEPAGE+158},       ! Stern davor anzeigen
  ELSE                                  ! sonst ein Leerzeichen
    PRINT ' CHAR{BASEPAGE+158},         ! vor den Dateinamen
  ENDIF
  e%=FSNEXT()                           ! Suche fortsetzen
LOOP
```

--> En la pantalla se visualizan todos los archivos con el sufijo .GFA provenientes del índice actual, así como todos los nombres de ordenadores, que también poseen ese sufijo, marcados con *.

MKDIRnombre$
RMDIRnombre$

nombre$: sexp

La instrucción MKDIR(make directory) presenta un índice (ordenador). La expresión string nombre$ contiene el correspondiente camino de acceso. Con RMDIR (remove directory) se borra un índice (ordenador), pero sólo cuando este índice no contiene subíndices o archivos.

Ejemplo

```
MKDIR "A:\TEXTVER"
RMDIR "A:\TEXTVER"
```

-->En la primera línea se instala un índice "TEXTVER" en la disquetera A, que se vuelve a borrar en la segunda línea.

Archivos

EXIST(nombre$)

nombre$: sexp

Con ayuda de EXIST se puede determinar, si existe un archivo. Para ello se indica en la expresión string nombre$ el camino de acceso de este archivo. La función retorna TRUE(-1), cuando el archivo existe, sino da FALSE(0) (comparar: FSFIRST y FSNEXT).

Ejemplo

```
OPEN "U",#1,"TEST.TXT"
PRINT #1,"BEISPIEL"
CLOSE #1
PRINT EXIST("TEST.TXT")
PRINT EXIST("TEST.DOC")
```

--> En las dos primeras líneas se abre un archivo con el nombre "TEST.TXT". La cuarta línea revisa la existencia del archivo "TEST.TXT", se retorna TRUE. En la última línea la revisión da FALSE, ya que no existe el archivo. Comparar también: FSFIRST, FSNEXT.

OPENmodus$,#n,nombre$[anz]

modus$,nombre$: sexp
anz,n: iexp

OPEN abre un canal de datos hacia un archivo o un aparato periférico. En este caso la expresión string 'modus$' determina una de las siguientes posibilidades de acceso:

O	(output)	Abre un archivo para escribir. Para ello eventualmente se organiza de nuevo o bien se borran datos exisentes.
I	(input)	Abre un archivo para leer.
A	(append)	Hace posible anexar datos a un archivo existente. En este caso el indicador de datos está dirigido al final del archivo.
U	(update)	Abre un archivo existente para leer y escribir.
R	(random access)	Abre un archivo para escribir y leer a elección. En la instrucción FIELD se da una descripción de este tipo de archivos.

La expresión numérica n contiene el número de canal y puede asumir valores de 0 a 99. Este número de canal se debe indicar cuando se quiere trabajar con el archivo. En este caso se puede suprimir el símbolo # antes de la indicación del canal. La expresión string nombre$ contiene el camino de acceso del archivo. Como nombre del archivo también se puede indicar una caracterización de periféricos (ver lista en INP). La expresión numérica anz solo se evalúa en archivos de libre acceso (random access); anz contiene la longitud de una oración de datos.

Abreviaturas		Significado	Uso interno
LST:	(list)	paralelo	BIOS 0
AUX:	(auxiliar)	en serie (RS 232)	BIOS 1
CON:	(consola)	Teclado/monitor	BIOS 2 o VDI
MID:	(musical instruments digital interface)	Interfase MIDI	BIOS 3
IKB:	(intelligent keyboard)	Procesador del teclado	BIOS 4
VID:	(video)	Monitor	BIOS 5 o VDI
PRN:	(printer)	Impresora	Gemdos - 3

LOF(#n)
LOC(#n)
EOF(#n)
CLOSE[#n]
TOUCH[#]n

anz, n: iexp

Las funciones LOF (lenght of archivo), LOC (location) y EOF (end of archivo) solo se pueden aplicar en archivos previamente abiertos. Las tres funciones tienen la expresión numérica "n" en común. Indica el número de canal del archivo, al que se refiere la correspondiente función. LOF da la longitud de un archivo en bytes. LOC da posición actual del indicador de datos también en bytes (ver también SEEK). EOF indica, si el indicador de datos señala el final del archivo (o bien si el archivo se ha leído completamente). Cuando el indicador de datos señala al final del archivo, se retorna TRUE (-1), caso contrario FALSE (0).

CLOSE cierra un canal de archivo previamente abierto con OPEN en un archivo o en un aparato periférico. En este caso, la expresión numérica "n" contiene el número del canal a cerrar. Sin una indicación especial del canal, se cierran todos los archivos abiertos.

TOUCH actualiza la hora de un archivo, o sea el registro de la fecha y de la hora de un archivo abierto se ajusta a los datos del sistema.

Ejemplos

```
OPEN "O",#1,"test.txt"
FOR i%=1 TO 20
  PRINT #1,STR$(i%)
NEXT i%
CLOSE #1
FILES "test.txt"
DELAY 20                ! 20 segundos de pausa
OPEN "u",#1,"test.txt"
TOUCH #1
CLOSE #1
FILES "test.txt
```

--> En el ejemplo se abre y cierra para la lectura el archivo "test.txt" bajo el número de canal 1. A continuación se visualizan en la pantalla los datos de este archivo, se actualizan después de 20 segundos y se vuelven a visualizar.

```
OPEN "I",#1,"test.txt"
PRINT "   Largo del archivo: ";LOF(#1)
PRINT
PRINT "   Datos","Posicion del indicador"
DO UNTIL EOF(#1)
  INPUT #1,a$
  PRINT "   ";a$,LOC(#1)
LOOP
CLOSE #1
```

--> El ejemplo abre el mismo archivo para la lectura. Primero sevisualiza la longitud del archivo mediante LOF. A continuación, se indican en la pantalla el contenido del archivo y la correspondiente posición del indicador de datos. La condición para la interrupción del loop se fija con EOF. La lectura de los datos ocurre(independientemente de la longitud del archivo calculada anteriormente) por revisión del indicador de datos. En este caso se vuelve a ejecutar tantas veces el loop, hasta que la función EOF dé el valor TRUE (-1).

NAMEviejo$ASnuevo$
RENAMEviejo$ASnuevo$
KILLnombre$

viejo$,nuevo$,nombre$: sexp

NAME cambia el nombre de un archivo con el camino de acceso viejo$ por el nombre del archivo con el camino de acceso nuevo$. El contenido del archivo no se modifica. Los archivos viejo$ y nuevo$ deben hallarse en la misma disquetera. RENAME equivale a la instrucción NAME.

KILL borra un archivo, cuyo camino de acceso se indica en la expresión string "nombre$".

Ejemplo

```
OPEN "O",#1,"prueba.txt"
PRINT #1,"Ejemplo"
CLOSE #1
'
NAME "prueba.txt" AS "ejemplo.txt"
DIR
KILL "ejemplo.txt"
DIR
```

--> En el primer ejemplo se abre el archivo "test.txt" para escribir. A continuación, se cambia el nombre de este archivo por "ejemplo.txt" y finalmente se borra. Esto se controla a través de la salida del índice. Después de NAME o de KILL se visualizan en la pantalla los índices modificados.

BLOADn$[,adr]
BSAVEn$,adr,anz

nombre$: sexp
n,adr,anz: aexp

Con BSAVE se puede almacenar una zona de la memoria sobre el disco (RAM-disk, disco rígido, etc.), el cual se puede volver a cargar por

ej. con BLOAD. La expresión numérica adr indica la dirección inicial de la zona de la memoria. Sin ninguna indicación de adr se usa la última dirección nombrada por BSAVE. En el caso de BSAVEse debe indicar además anz (la longitud del archivo n$). El parámetro n$ es el nombre del archivo a cargar o a almacenar. Para n$ valen las reglas descriptas bajo DIR del sistema jerárquico de archivos.

BSAVE,BLOAD sólo pueden solicitar archivos completos por su nombre. De lo contrario, BPUT y BGET solicitan archivos a través de su número de canal n. Así es posible cargar o almacenar partes de un archivo (inclusive varias veces) con BGET y BPUT.

Ejemplos

```
DEFFILL 1,2,4
PBOX 100,100,200,200
BSAVE "rect.pic",XBIOS(2),32000
CLS
PRINT AT(4,20);"Ha sido guardado. Pulse una tecla!"
~INP(2)
CLS
BLOAD "rect.pic"
```

--> Dibuja un rectángulo y almacena la pantalla bajo "rectangulo.pic". Entonces aparece el correspondiente mensaje. Después de una pulsación de tecla se vuelve a cargar el archivo almacenado en la pantalla.

```
DEFFILL 1,2,4
PBOX 0,0,639,199
DEFFILL 1,2,2
PBOX 0,200,639,399
DEFTEXT 1,0,0,32
TEXT 100,115,"La mitad de arriba"
TEXT 100,315,"La mitad de abajo"
'
OPEN "O",#1,"screen.pic"
BPUT #1,XBIOS(2),32000
CLOSE #1
PAUSE 25
'
OPEN "I",#1,"screen.pic"
BGET #1,XBIOS(2)+16000,16000
BGET #1,XBIOS(2),16000
CLOSE #1
```

--> Rellena la mitad superior del monitor con una muestra y la mitad inferior con otra. Luego se almacena toda la pantalla en el archivo screen.pic y se vuelve a leer después de una pausa. En este caso se carga la primera mitad del archivo en la mitad inferior de la pantalla y la segunda mitad en la mitad superior de la pantalla.

Acceso secuencial

```
INP(#n)
OUT#n,a[,b,c,...]
```

n,a,b,c,...: iexp

INP(#n) lee un byte de un archivo abierto anteriormente. En forma análoga transfiere OUT un byte a un archivo. La expresión numérica "n" contiene en este caso el número de canal (0 a 99), bajo el cual se solicita el correspondiente archivo. INP y OUT sin "#" se pueden usar también para la comunicación con el monitor, IKBD, la interfase MIDI, etc. (ver allí).

Con OUT se emplea BYTE(a).

Ejemplo

```
OPEN "O",#1,"prueba.txt"
OUT #1,128
CLOSE #1
'
OPEN "I",#1,"prueba.txt"
a=INP(#1)
CLOSE #1
PRINT a
```

--> En el primer ejemplo se abre un archivo para escribir y se escribe un byte, que se escribe en la segunda parte del ejemplo en la variable numérica "a" y se visualiza en la pantalla. En este caso "a" tiene el valor 128.

INPUT$(anz[,#n])

n,anz: iexp

INPUT$ lee una cadena de caracteres compuesta por anz caracteres del teclado. La indicación de un número de canal "n" (0 a 99) produce opcionalmente la lectura de caracteres provenientes de un archivo. En ambos casos la expresión numérica "anz" indica, cuántos caracteres se han de leer.

Ejemplo

```
OPEN "O",#1,"version.dat"
PRINT #1,"GFA-BASIC, Version 3.0"
CLOSE #1
'
OPEN "I",#1,"version.dat"
v$=INPUT$(9,#1)
CLOSE #1
PRINT v$
PRINT "Numero de version: ";
PRINT INPUT$(3)
```

--> En el primer ejemplo se abre el archivo "version.dat" y se escribe un mensaje en el archivo. La segunda parte del ejemplo lee los primeros nueve caracteres de este archivo en la variable string v$ y la visualiza luego en la pantalla. Entonces aparece un mensaje y se leen tres caracteres del teclado y se visualizan en el monitor.

INPUT#n,var1[,var2,var3,...]
LINE INPUT#n,a1$[,a2$,a2$,...]

n: iexp
a1$,a2$,a3$: sexp
var1,var2,var3: avar o svar

INPUT #n hace posible leer datos de un archivo. En este caso se pueden leer valores individuales o listas de variables, donde las variables están separadas por comas. Estas instrucciones equivalen a INPUT o INPUT LINE, solo que aquí (en la mayor parte de los casos) no se lee desde el teclado.

Ejemplo

```
OPEN "I",#1,"TEXT.DOC"
INPUT #1,a$,b$
LINE INPUT #1,c$
CLOSE #1
PRINT a$
PRINT b$
PRINT c$
```

--> Lee tres strings de un archivo (que se debe abrir anteriormente) y los retorna en la pantalla.

PRINT#n,expresión
PRINT#n,USINGforma$,expresión
WRITE#n,expresión

n: iexp
forma$: sexp
expresión: aexp o sexp o combinaciones de éstos

PRINT#n visualiza datos en un canal de datos. PRINT#n,USING permite la salida formateada en un canal de datos. En ambos casos n representa el número de canal (0 a 99) del correspondiente archivo. En lo demás es igual que PRINT, PRINT USING y WRITE. Sin embargo, no es posible PRINT#n,AT(,).

La instrucción WRITE sirve en primer lugar para almacenar datos en archivos secuenciales ahorrando espacio y está configurado para la lectura de archivos con la instrucción INPUT. Las expresiones se separan con comas y las cadenas de caracteres se deben figurar entre comillas.

Ejemplo

```
OPEN "O",#1,"TEXT.DOC"
PRINT #1,"Prueba",a$
PRINT #1,"GFA-","BASIC"
CLOSE #1
```

--> Lee tres strings de un archivo (que se debe instalar previamente) y los retorna en la pantalla.

```
OPEN "O",#1,"TEST.DAT"
WRITE #1,"Version",3,".0"
CLOSE #1
```

```
'
OPEN "I",#1,"TEST.DAT"
INPUT #1,v1$,v2$,v3$
CLOSE #1
'
PRINT v1$+v2$+v3$
```

--> Escribe datos separados con comas en el archivo TEST.DAT, a continuación vuelve a leer los datos con INPUT y los retorna en la pantalla.

STORE#i,x$()[,n]
RECALL#i,x$(),n,x

i,n: iexp
x$(): arreglo string
x: variable de por lo menos 32 bits

La instrucción STORE sirve para almacenar un campo de strings como archivo de texto (separado por CR/LF). Se visualiza el arreglo string completo a través del canal abierto i. El parámetro de libre elección n puede indicar, cuántos elementos del arreglo string se han de visualizar.

La instrucción RECALL sirve para la lectura rápida de un campo string de un archivo de texto. Se leen n líneas del archivo de texto en el arreglo string. Si n es demasiado grande para dimensionar el arreglo string, entonces se limita automáticamente la cantidad que se debe leer (=-1 lee todo el arreglo). Si durante la lectura se alcanza el final del archivo (EOF), se interrumpe la lectura sin mensaje de error. Finalmente la variable x siempre contiene la cantidad de strings realmente leídos.

Ejemplos

```
DIM A$(1000)
FOR i%=0 TO 499
a$(i%)=STR$(RND)     !cualquier cosa
NEXT i%
OPEN "O",#1,"testfile.txt"
STORE #1,a$(),500
CLOSE #1
'
DIM b$(2000)
OPEN "I",#1,"testfile.txt"
RECALL #1,b$(),-1,n
CLOSE #1
PRINT n
```

--> Se visualiza la cantidad de líneas del texto leídas, en este caso 500.

```
PRINT "Zeilenzählen"
DIM a$(1000)
DO
  FILESELECT "\*.*","",f$
  EXIT IF f$=""
  lc%=0
  OPEN "I",#1,f$
```

```
  DO
    RECALL #1,a$(),-1,x%
    ADD lc%,x%
  LOOP WHILE x%
  CLOSE #1
  PRINT f$;" contiene ";lc%;" lineas"
LOOP
```

--> Este programa cuenta las líneas en los archivos de texto.

Nota: STORE también funciona para archivos orientados con caracteres (AUX:), en cambio RECALL no, ya que internamente se emplea un SEEK.

SEEK#n,pos
RELSEEK#n,anz

n,anz,pos: iexp

Las instrucciones SEEK y RELSEEK permiten posicionar el indicador de datos. De esta forma se puede realizar un acceso a datos secuenciales. En este caso la expresión numérica "n" contiene el número de canal de un archivo abierto previamente con OPEN. Ambas instrucciones solo se pueden emplear con archivos, no con aparatos periféricos. El indicador de datos señala, qué byte de un archivo ha sido leído o escrito como último . Excepto en el modo "A" (inserción de datos en un archivo existente) el indicador de datos tiene el valor 0 al abrirse un archivo. Las instrucciones de escritura y lectura comienzan con el primer byte, que señala el indicador de datos.

Para la posición absoluta del indicador de datos se usan las instrucciones SEEK. Para ello se apunta el indicador de datos sobre el byte indicado en "pos". La ubicación relativa del indicador se realiza con la instrucción RELSEEK. Aquí se corre el indicador de datos desde su posición actual por un valor indicado en "anz". RELSEEK generalmente es más rápido!

Las expresiones numéricas "anz" y "pos" solo pueden asumir valores entre 0 y la correspondiente longitud del archivo. El indicador de datos se desplaza en dirección al final del archivo con valores positivos y en dirección al comienzo del archivo con valores negativos. Cuando se indica 0, el indicador de datos señala el comienzo del archivo.

Ejemplo

```
OPEN "O",#1,"X.X"
PRINT #1,STRING$(20,42)
SEEK #1,10
PRINT #1,"#";
RELSEEK #1,-5
OUT #1,33,48
CLOSE #1
OPEN "I",#1,"X.X"
LINE INPUT #1,a$
PRINT a$
CLOSE #1
```

--> Se visualiza: ******!0**#********

Archivo aleatorio

A continuación se explica el manejo de los llamados archivos con random-access (archivo aleatorio). Aquí tienen mucha importancia los siguientes dos conceptos: registro tipo de un archivo y campo de datos. El registro tipo de un archivo representa de aquí en más un resumen lógico de datos, por ej. el registro tipo "direcciones". Este registro tipo de un archivo puede estar subdividido en campos de datos, por ej. "nombre", "calle". En forma análoga se usan los conceptos longitud del registro tipo de un archivo y del campo; ellos indican la longitud del registro tipo o bien del campo de datos en bytes.

La diferencia fundamental entre un archivo random o aleatorio y un archivo secuencial consiste en el acceso de los datos. En un archivo secuencial se debe cargar todo el archivo, para poder tener acceso a un registro tipo de un archivo. En cambio, un registro tipo se puede leer de un archivo random sin tener que leer todo el archivo. Esto es muy útil cuando se trata de archivos muy extensos. Sin embargo, esta ventaja trae aparejado un mayor requerimiento de espacio de la memoria del disco, ya que un archivo aleatorio o random trabaja con longitudes de registro tipo y de campo fijas. Estas longitudes del registro tipo y de campo se almacenan independientemente de la longitud de los datos.

FIELD#n,anzASregistro$[,anzASregistro$,anzASregistro$,...]
FIELD#n,anzAT(x)[,anzAT(x)[,anzAT(x)[,....]

n,anz,x: iexp
registro$: svar, pero no variable de campo

La instrucción FIELD AS subdivide registros de datos en campos. La expresión numérica n es el número (de 0 a 99) bajo el cual fue abierto un archivo random en la instrucción OPEN correspondiente. La expresión entera anz determina la longitud del campo. La variable string registro$ contendrá un campo del registro tipo de un archivo. Si el registro se debe subdividir en varios campos, se deben separar las partes de las instrucciones (anz AS registro$) con comas. La suma de las longitudes de campo individuales debe equivaler a la longitud del registro, sino se imprimirá en pantalla un mensaje de error. Para que los distintos registros no tengan una longitud diferente a la fijada con la instrucción FIELD, resulta conveniente usar las instrucciones LSET y RSET o MID$.

Con AT se pueden ingresar por ej. variables numéricas en un archivo aleatorio (random access), sin que se deban transferir a cadenas de caracteres, si entre los paréntesis figura un indicador que señale la variable numérica a almacenar y delante de AT figura la cantidad de bytes, que se deben leer y almacenar a partir de esta dirección. También se pueden indicar otras direcciones Por ej.:

```
FIELD #1,4 AT(*a%),2 AT(*b&),8 AT(*c#)
```

Además son posibles combinaciones de AS y AT, por ej.:

```
FIELD #2,AS a$,2 AT(*b&),8 AT(*c#),6 AS d$
```

A diferencia de la versión 2.0 se puede subdividir una instrucción FIELD en varias líneas. La longitud máxima del registro vale 32767, la

cantidad máxima de campos vale alrededor de 5000.

GET#n[,registro]
PUT#n[,registro]
RECORD#n,registro

n,registro: iexp

GET lee un registro de un archivo aleatorio (random). PUT almacena en forma análoga un registro en un archivo de este tipo. En este caso n es el número (0 a 99) bajo el cual fue abierto el archivo random en la correspondiente instrucción OPEN. El parámetro opcional registro contiene un valor entre 1 y la cantidad de registros dentro del correspondiente archivo. Este indica el número del registro a leer o almacenar. Si no se indica registro se lee o almacena el próximo registro correspondiente.

Con RECORD sólo se fija el siguiente número de registro para PUT o GET.Después de RECORD #1,15, GET #1 lee por lo tanto el registro Nro.15 .

Atención: Un archivo siempre se puede alargar por un registro más, eventualmente cuando se incorpora dentro de un loop se pueden agregar varios registros.

Ejemplo

```
OPEN "R",#1,"\privado.rdm",62
FIELD #1,24 AS nombre$,24 AS calle$,2 AT(*cod&),12 AS lugar$
'
FOR i%=1 TO 3
INPUT "Nombre   : ";n$
INPUT "Calle: ";s$
INPUT "Codigo postal   : ";cod&
INPUT "Lugar    : ";o$
LSET nombre$=n$
LSET calle$=s$
LSET lugar$=o$
PUT #1,i%
CLS
NEXT i%
'
CLOSE #1
```

--> En primer lugar se abre un archivo random (modo "R") con una longitud de registro de 62 bytes. Con la instrucción FIELD se subdivide un registro en 2 campos de datos con cada uno 24 bytes, un campo con 2 bytes y un campo de datos con 12 bytes. La suma de los campos (24+24+2+12) da nuevamente 62 (o sea la longitud de registro indicada en la instrucción OPEN). A continuación se ingresan los datos de direcciones y se incorporan a la izquierda de las correspondientes variables. Entonces se escribe todo el registro en el archivo. En versiones anteriores de GFA-BASIC diría 2 AS cod$ en lugar de 2AT(*cod%) y luego delante de PUT: LSET cod$=MKI$(cod%) y en forma equivalente detrás de GET:cod%=CVI(cod$).

Ejemplo

```
OPEN "R",#1,"\privat.rdm",62
FIELD #1,24 AS nombre$,24 AS callee$,2 AT(*cod&),12 AS lugar$
'
FOR i%=1 TO 3
  GET #1,i%
  PRINT "Numero de datos: ";i%
  PRINT "Name    : ";nombre$
  PRINT "Strasse: ";calle$
  PRINT "PLZ     : ";cod&
  PRINT "Ort     : ";lugar$
NEXT i%
'
CLOSE #1
```

--> Aquí se abre el archivo aleatorio "PRIVAT_1.RDM"
y se leen 3 registros.

Comunicación con los periféricos

Entrada y salida byte a byte

INP(n)
INP?(n)
OUT[#]n,a[,b...]
OUT?(n)

n,a,b: iexp

INP lee un byte de los periféricos. La expresión numérica "n" puede asumir valores entre 0 y 5 (ver tabla más adelante) o puede contener un número de canal. La instrucción OUT transfiere bytes a los periféricos. Contrariamente a versiones 2.xx del GFA-BASIC ahora se pueden transferir varios bytes a los periféricos o bien a un archivo.

INP? o OUT? refleja el estado de entrada o de salida de un periférico. En este caso se retorna un número distinto de cero, el aparato indicado en n está listo, sino se retorna FALSE(0).

n	Abreviatura		Significado
0	LST:	(list)	Impresora
1	AUX:	(auxiliar)	en serie (RS 232)
2	CON:	(consola)	teclado
3	MID:	(musical instruments digital interface)	Interfase MIDI
4	IKB:	(intelligent keyboard)	Procesador del teclado
5	VID:	(video)	Monitor

Ejemplos

```
PRINT AT(4,4);"Oprima una tecla!"
~INP(2)
```

--> En la pantalla se visualiza un mensaje y espera que se pulse una tecla (número de periférico 2, teclado).

```
OUT 2,27,69,13
```

--> Equivale a la instrucción CLS, borra la pantalla y fija el cursoren el extremo izquierdo superior con ayuda de VT-52-Sequenz Clear Screen (esc+"E").

Interfase seriada y MIDI

INPAUX$
INPMID$

Con ayuda de INPAUX$ y de INPMID$ se pueden leer con gran velocidad datos a través del interfase serie y MIDI.

Ejemplos

```
DO
  PRINT INPAUX$;
LOOP UNTIL MOUSEK
```

--> Lee con gran rapidez todos los datos del buffer INPUT del interfase serie. Esta forma de ingresar datos es mucho más veloz que la lectura byte a byte a través del interfase.

```
inp_aux$ = ""
WHILE INP?(1)
  inp_aux$ = inp_aux$ + CHR$(INP(1))
WEND
```

--> Esta variante lee los datos marcadamente más despacio.

Ratón y joystick

MOUSEX
MOUSEY
MOUSEK
MOUSExmx,my,mk

mx,my,mk: avar

Las instrucciones MOUSEX, MOUSEY y MOUSEK hacen posible solicitar la posición actual del ratón y el estado del teclado del ratón. Con la instrucción MOUSE se pueden resumir estas solicitudes. MOUSE da entonces las coordenadas actuales del ratón en mx y my y en mk el estado del teclado del ratón. La variable mk puede asumir en este caso los valores 0 y 3.

mk	Tecla pulsada
0	Ninguna
1	Izquierda
2	Derecha
3	Izquierda y derecha

Ejemplo

```
REPEAT
  IF MOUSEK=1
    PLOT MOUSEX,MOUSEY
```

```
  ENDIF
UNTIL MOUSEK=2
'
REPEAT
  MOUSE mx%,my%,mk%
  IF mk%=2
    PLOT mx%,my%
  ENDIF
UNTIL mk%=1
```

--> En el loop REPEAT se solicitan posición del ratón y estatus del teclado del ratón. Solo mientras se pulsa la tecla izquierda del ratón, se dibuja un punto la posición mx,my. El loop se abandona después de pulsar la tecla derecha. En el segundo loop se puede dibujar entonces con la tecla derecha del ratón y abandonar el programa con la tecla izquierda.

SETMOUSEmx,my[,mk]

mx,my,mk: iexp

La instrucción SETMOUSE permite posicionar el cursor del ratón a través del manejo del programa. Con el parámetro opcional mk se puede simular pulsar o dejar de pulsar una tecla del ratón. Esto vale lamentablemente solo para el VDI, y no (o solo raras veces) para el AES.

Ejemplo

```
FOR i%=00 TO 300
  SETMOUSE i%,i%
  PAUSE 2
  PLOT MOUSEX,MOUSEY
  SHOWM
NEXT i%
```

--> Desplaza el cursor del ratón en forma diagonal a lo largo de la pantalla fijando puntos.

HIDEM
SHOWM

Los comandos HIDEM y SHOWM conectan o desconectan el cursor del ratón. Cuando se llama un comando Alert u otro de la rutina AES se conecta al igual que con las instrucciones que solicitan el ratón.

En la salida por el monitor (PRINT) o en instrucciones de gráficos (VDI,LINE-A) se desconecta el ratón.

Con HIDEM se puede bloquear por movimiento (entre instrucciones) la reconexión automática del cursor del ratón.

Ejemplo

```
REPEAT
  IF MOUSEK=1
    SHOWM
  ENDIF
```

```
  IF MOUSEK=2
    HIDEM
  ENDIF
UNTIL MOUSEK=3
```

--> Pulsando la tecla izquierda del ratón se señala el cursor del ratón, después de pulsar la tecla derecha desaparece de la pantalla. Si se pulsan ambas teclas simultáneamente se finaliza el programa.

STICK(p)
STRIG(p)

n,p: iexp

El ordenador ATARI ST dispone de dos interfases (ports) para la conexión del ratón y del joystick. En el port 0 se pueden solicitar coordenadas del ratón y del joystick, en el port 1 solo se pueden solicitar datos del joystick.

La instrucción STICK 0 conecta un modo, en el cual se retornan coordenadas el ratón desde el port 0. Para STICK 1 se retornan coordenadas del ratón desde Port 0 y 1. Normalmente no es necesario el empleo de la instrucción STICK, ya que cuando se solicitan las coordenadas del ratón se ejecuta automáticamente un STICK 0 y cuando se solicita el joystick el STICK 1. En programas que solicitan joystick debería preceder a cada llamado de funciones AES (Alert-boxes,Dialogboxes) un STICK 0.

La función STICK(p) sirve para solicitar la posición del joystick. Para p=0 se reporta la posición del joystick en Port 0, para p=1 la del joystick en Port 1.

```
    5  1  9
     \ | /
  4 -- 0 -- 8
     / | \
    6  2  10
```

De acuerdo con esto, String(p) indica el estatus del botón de disparo (TRUE=pulsado,sino FALSE).

Ejemplos

```
STICK 1
'
REPEAT
  direccion%=STICK(1)
  fuego!=STRIG(1)
  SELECT direccion%
  CASE 4
    PRINT "izquierda"
  CASE 8
    PRINT "derecha"
  CASE 2
    PRINT "abajo"
  CASE 1
```

```
    PRINT "arriba"
  ENDSELECT
UNTIL fuego!
WHILE STRIG(1)
WEND                    ! Espera a que se deje de pulsar el boton
```

--> Al cambiar al ratón, las informaciones del Joystick son anuladas, o sea que el ordenador no nota que los botones de fuego son pusados. Esto puede ser resuelto como en el ejemplo esperando que se deje de pulsar de ambos botones o informando al programa sobre la alteracion.

```
PRINT "Oprima el boton de fuego"
WHILE STRIG(0)
WEND
REPEAT
UNTIL STRIG(0)
WHILE STRIG(0)
WEND
```

--> Parte del ejemplo anterior.

La impresora

LPRINTexpresión
LPOS(x)
HARDCOPY

expresión: aexp o sexp o combinación de ellos
x: avar, argumento ficticio

La expresión LPRINT hace posible la salida de datos a la impresora. En este caso LPRINT es idéntico a la instrucción PRINT. Básicamente se pueden emplear todas las variaciones de la instrucción PRINT. Pero no es posible posicionar con PRINT AT(x,y) la cabeza de impresión. Con PRINT USING ésto sí es posible.

LPOS(x) indica la cantidad de caracteres impresos a partir del último CR en la impresora (comparar POS(x)).

HARDCOPY produce la salida del contenido de la pantalla en una impresora adecuada. Aquí se llama la rutina hardcopy del sistema operativo. Esta rutina también se puede llamar pulsando simultáneamente las teclas Alternate y Help. Hay programas driver para impresoras Epson-no compatibles de 9 agujas, que no reaccionan a la instrucción HARDCOPY (XBIOS(20)). En este caso sirve SDPOKE#H4EE,0.

Ejemplos

```
LPRINT
LPRINT "Prueba"
PRINT LPOS(x)
```

--> En la impresora (si está anexada y conectada) se produce un avance a la próxima línea, escribe la palabra "Test" en la impresora e indica la posición actual de la cabeza de impresión en la pantalla.

```
FOR i%=20 TO 180 STEP 10
  CIRCLE 320,200,i%
NEXT i%
HARDCOPY
```

--> Dibuja círculos concéntricos en la pantalla y se visualiza este dibujo en la impresora.

Generación de sonido

SOUNDkan,laut,nota,octava,verz
SOUNDkan,laut,#per,verz
WAVEstim,huell,form,duración,verz

kan,laut,nota,octava,verz,per,stim,huell,form,duración: iexp

Con las instrucciones SOUND y WAVE se maneja el generador de tono de tres voces del ATARI ST. Para ello se usa en SOUND como primer parámetro la expresión "kan";que indica el canal deseado (1 a 3). El volumen (1 a 5) se maneja a través del segundo parámetro "laut". El parámetro "nota" y "octava" sirven para fijar la nota (1 a 12) en la zona de las octavas (1 a 8). En este caso vale:

Nota	I	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
Número	I	1	2	3	4	5	6	7	8	9	10	11	12

Una variante de la instrucción SOUND permite indicar un período de ondas mediante la expresión numérica "per", al cual debe preceder el símbolo #. En este caso "per" puede asumir los valores entre 0 y 4095 y se puede calcular de la siguiente manera, si se conoce la frecuencia en Hertz:

```
per=ROUND(125000/frecuencia)
```

Así se puede generar de dos maneras diferentes el tono del cánal A (440 Hz) :

```
SOUND 1,15,10,4,250
SOUND 1,15,#284,250
SOUND 1,0,0,0,0
```

En ambas variantes de la instrucción sigue un último parámetro "verz"; indica el tiempo en 1/50 segundos hasta la siguiente ejecución de la instrucción (5 segundos en el ejemplo). La generación de sonido se finaliza con la instrucción SOUND 1,0,0,0,0 (como en el ejemplo) o con cualquier otra instrucción SOUND, como lo produce por ej. el sistema operativo al pulsarse una tecla. Esta pulsación de tecla se puede desconectar con:

```
SPOKE &H484, BCLR(PEEK(&H484),0)
```

Se puede conectar con:

```
SPOKE &H484, BSET(PEEK(&H484),0)
```

La generación de tonos de varias voces se logra con WAVE. En "stim" se suman los valores bit de las tres voces y se fija la frecuencia del generador de susurro:

1= Voz 1
2= Voz 2
4= Voz 3
8= Susurro para la primera voz.
16= Susurro para la segunda voz.
32= Susurro para la tercera voz.

A esto se puede sumar 256 veces el período del generador de susurrp (0 a 31). En "huell" se indica bit a bit igual que en "stim", qué voz se ha de modular con una curva de envoltura. La forma de la curva de envoltura se puede determinar con "form" de la siguiente manera:

0 a 3 = cae linearmente.
4 a 7 = crece linearmente, se interrumpe.
8 = cae en zig-zag
9 = cae linearmente (como 0 a 3).
10 = triángulo, al comienzo cayendo.
11 = cae linearmente, esporádicamente fuerte.
12 = crece en zig-zag.
13 = crece linearmente, se mantiene.
14 = triángulo, crece al comienzo.
15 = crece linearmente, se interrumpe (como 4 a 7).

El período de la curva de envoltura se determina con "duración". Como en la instrucción SOUND, "verz" fija el tiempo antes de la próxima ejecución de la instrucción en 1/15 segundos.

Para SOUND y WAVE vale: Se pueden suprimir los parámetros que figuran al final de la instrucción y que no se deben modificar.

Ejemplo

```
SOUND 1,15,1,4,20
SOUND 2,15,4,4,20
SOUND 3,15,8,4,20
SOUND 7,7,0,65535,300
```

--> Con cada canal de tono se genera un tono y se modula con ayuda de WAVE.

7 - Control del programa

En este capítulo se discuten todas las instrucciones para el control de un programa. Comienza con las instrucciones de desición que posibilitan una ejecución condicional de instrucciones (IF,THEN,ELSE,ENDIF,ELSE IF). En estas instrucciones se analiza únicamente, si una determinada condición lógica es verdadera o falsa.

Además existen las instrucciones de desiciones múltiples, emparentadas con las anteriormente mencionadas (SELECT,CASE,DEFAULT,ENDSELECT, CONT). En estas instrucciones el criterio de desición no solo puede ser verdadero o falso sino que también puede asumir valores cualesquiera, frente a los cuales puede reaccionar selectivamente.

El siguiente apartado presenta las instrucciones de estructuras de repeticón (loops) del GFA-BASIC, las cuales posibilitan la ejecución repetida de ciertas partes del programa. El GFA-BASIC3.0 dispone de una gran cantidad de estos tipos de loops (FOR TO STEP NEXT, REPEAT UNTIL, DO LOOP o bien ENDLOOP, DO WHILE, DO UNTIL, LOOP WHILE, LOOP UNTIL, EXIT IF).

Para la programación estructurada es muy importante la posibilidad de formación de subprogramas. Con las instrucciones PROCEDURE, GOSUB (@) y RETURN o bien ENDSUB se pueden crear funciones nuevas en forma de subrutinas. También es posible la creación de funciones. Con las instrucciones DEFFN y FN se pueden diseñar funciones, que tienen el carácter de fórmulas. Otras posibilidades aún más flexibles resultan de las instrucciones FUNCTION,ENDFUNC y RESULT, con las cuales se pueden crear subprogramas completos, que devuelven un valor.

En este apartado también se trata la administración de variables en subrutinas. Aquí existe en primer lugar la posibilidad de unificar variables locales (LOCAL), que solo son conocidas en una parte del programa. Las variables se pueden transferir a subrutinas en calidad de valor (call by value). Con la instrucción VAR también se pueden transferir por sí mismas (call by reference), de tal manera que la subrutina las puede modificar, sin que se deba usar su nombre global de variable.

En el apartado sobre ramificaciones condicionadas a un suceso se trata de dos sucesos específicos del GFA-BASIC. El primero es la pulsación simultánea de las teclas Control-, Alternate- y de la tecla Shift izquierda, lo que normalmente lleva a la interrupción del programa. El segundo suceso es la aparición de un error en la ejecución del programa. Aquí se analizan las posibilidades para la programación de una rutina de manjo de errores.

En el GFA-BASIC 3.0 es posible llamar una subrutina después de transcurrido un determinado tiempo. Con este fin se usan las instrucciones EVERY y AFTER.

Al final de este capítulo se presentan la bifurcación incondicional GOTO, las instrucciones de espera PAUSE y DELAY y diferentes posibilidades de interrupción del programa (QUIT, SYSTEM, END, EDIT, NEW, STOP).

El último apartado trata de las instrucciones para la supervisión de la ejecución del programa (TRON, TROFF, TRON proc, TRACE$ DUMP, ERR$,ERROR).

Instrucciones de desición

```
IFbed[THEN]
ELSE
ENDIF
```

bed: bexp

Con estos comandos se activan una o varias instrucciones siempre y cuando se cumpla una condición lógica. El siguiente ejemplo ilustrará ésto:

```
IF a=1 THEN
PRINT "a es igual a 1"
b=2
ENDIF
```

En este caso a=1 es la condición lógica. Las instrucciones en las líneas entre IF y ELSE sólo se ejecutan, cuando la condición lógica es cierta. Cuando no es cierta, se continua la ejecución del programa detrás de la palabra ENDIF de la instrucción. El comando THEN no se tiene en cuenta, o sea también es suficiente la siguiente formulación:

```
IF a=1    en lugar de IF a=1 THEN
```

La siguiente construcción es algo más compleja:

```
IF a=1
PRINT "a es igual a 1"
ELSE
PRINT " a no es igual a 1,"
PRINT " sino igual ";a
ENDIF
```

En este caso se ejecutan las instrucciones entre IF y ELSE, cuando la condición lógica detrás de IF es cierta. A continuación, el programa prosigue detrás del comando ENDIF.

Si no se cumplió la condición IF, entonces se activan las instrucciones entre ELSE y ENDIF. A continuación, se prosigue la ejecución del programa detrás de ENDIF. Cualquier expresión distinta de cero es lógicamente verdadera.

Ejemplo:

```
x=1
IF x
  PRINT "x es verdadero"
ENDIF
INPUT y
IF x=9 OR ODD(y)
  PRINT "y es un número impar"
ELSE
  PRINT "y es un número par"
ENDIF
```

--> Primero aparece el texto 'x es verdadero'. Luego se solicita un número. Ya que x no puede ser 9, se visualiza el texto 'y es un número impar', si Usted ha ingresado un número impar. De lo contrario se

escribe 'y es un número par' sobre el monitor.

ELSE IFbed

bed: bexp

La instrucción ELSE IF hace posible representar con mayor claridad instrucciones IF encasilladas. El siguiente ejemplo reacciona a la pulsación de teclas. Con S,L o E se llama a la anotación que se usa para rutinas de almacenamiento, de carga o de ingreso de datos. En todos los demás casos aparece el texto 'Comando desconocido'. La versión encasillada de este ejemplo es la siguiente:

El empleo de ELSE IF produce un listado más corto con menos desplazamientos del texto :

La ejecución de tales estructuras del programa se lleva a cabo de acuerdo con el siguiente patrón:

Si se cumple la condición detrás de IF (aquí t$="L") entonces se ejecutan las instrucciones entre IF y el siguiente ELSE IF (aquí "Cargar" PRINT) y luego se bifurca detrás de la instrucción ENDIF.

Si una de las condiciones detrás de las instrucciones ELSE IF es verdadera, entonces se ejecutan todas las instrucciones hasta el próximo ELSE, ELSE IF o ENDIF (si no existe ELSE) y a continuación se salta a la instrucción ENDIF,

Si ni la condición detrás de IF ni una instrucción detrás de ELSE IF son verdaderas, entonces se ejecutan las instrucciones entre ELSE y ENDIF (en la medida en que exista ELSE).

Ejemplo: ver arriba.

Bifurcaciones múltiples

ONxGOSUBproc1,proc2,...

x: iexp
proc1, proc2: nombres de procedimientos, sin parámetros

Esta instrucción se bifurca hasta el x-simo procedimiento que figura en la lista detrás de GO-SUB . En este caso x es una expresión numérica, cuyos lugares detrás de la coma (en la medida en que existan) se ignoran. Cuando x es menor que 1 o mayor que la cantidad de nombres de procedimientos detrás de GOSUB, no se llama ninguna subrutina. Después de acceder a una subrutina se continúa el programa detrás de ON x GOSUB.

En estas instrucciones no se pueden dar parámetros a los procedimientos.

Ejemplo:

```
x=3
ON x GOSUB proc1,proc2,proc3
x=1
ON x+1 GOSUB proc1,proc2,proc3,proc4
```

--> Primero se llama el procedimiento proc3, luego el procedimiento proc2.

```
SELECTx
CASEy[TOz] o CASEy[,z,...]
CASE TOy
CASEyTO
DEFAULT
ENDSELECT
CONT
```

x,y,z: iexp o constante string con una longitud máxima de 4 caracteres

La instrucción SELECT hace posible la bifurcaión dependiendo del valor de la expresión numérica x. El siguiente ejemplo sirve para ilustrar la estructura del programa resultante:

Ejemplo:

```
x=0
SELECT x+2
CASE 1
  PRINT "x es igual a 1"
CASE 2 TO 4
  PRINT "x es igual a 2,3 o 4"
CASE 5,6
PRINT "x es igual a 5 o 6"
DEFAULT
  PRINT "x no es igual a 1,2,3,4,5 o 6"
ENDSELECT
```

--> En la pantalla se visualiza 'x es igual a 2,3 o 4'.

Primero se determina la expresión numérica detrás de SELECT, que es la condición de bifurcación (aquí igual a 2). Luego se revisan de arriba para abajo las instrucciones CASE y se comprueba si detrás de ellas se encuentra el valor actual de la condición de bifurcación. En este ejemplo figura solo 1 detrás de la primera instrucción CASE.Como la condición de bifurcación es igual 2, el programa salta al siguiente CASE.

Detrás del segundo CASE dice '2 TO 4'. Esta condición se satisface, cuando el criterio de bifurcación detrás de SELECT asume valores entre 2 y 4 (inclusive). En el caso anterior se produce entonces la ejecución de las instrucciones entre el segundo y tercer CASE. Luego se continúa la ejecución del programa detrás del comando END-SELECT.

Detrás del tercer CASE se puede reconocer otra variante de la indicación de los posibles valores del criterio SELECT. Los valores deseados ,separados por comas, se enumeran allí en una lista.

Si no está indicada la expresión actual del criterio de ramificación detrás de ningún CASE, entonces se ejecutan las instrucciones entre DEFAULT y END-SELECT, si existe una instrucción DEFAULT. En lugar de DEFAULT también se puede indicar OTHERWISE, lo que el intérprete reemplaza automáticamente por DEFAULT.

Detrás de CASE no solo pueden figurar constantes numéricas sino

también constantes String con una longitud máxima de cuatro caracteres. El string detrás de CASE puede contener como máximo cuatro caracteres en este caso. Cuando solo se indica un carácter se usa su valor ASCII como criterio de bifurcación. Si se indican dos caracteres, este valor se calcula de la siguiente manera:

Valor ASCII del primer carácter + 255 * valor ASCII del segundo carácter. De la misma manera se procede cuando se indican tres o cuatro caracteres.

Ejemplo:

```
raus!=FALSE
REPEAT
  taste%=inp(2)
  SELECT taste%
  CASE "a" TO "z"
  PRINT "Introdujo la letra pequeña "+chr$(taste%)+"."
  CASE "A" TO "Z"
    PRINT "Introdujo la letra grande "+chr$(taste%)+"."
  CASE 27
raus!=TRUE   !Fin de programa por medio de <Esc>
    DEFAULT
    PRINT "Pulsó una letra incorrecta !"
  ENDSELECT
UNTIL raus!
```

--> Dentro de una estructura de repetición se comprueba si ha sido activado el teclado, y de acuerdo con la tecla pulsada se ramifica o se reporta el correspondiente carácter o aparece el mensaje de un ingreso incorrecto de datos. Como criterio de interrupción del loop se pulsa la tecla Esc.

El siguiente ejemplo ilustra el significado de la instrucción CONT, la cual solo tiene importancia, si figura antes de una instrucción CASE o DEFAULT. Esta instrucción CONT no se debe confundir con el comando de igual nombre para volver a incorporar un programa interrumpido.

Ejemplo:

```
x=1
SELECT x
CASE 1
  PRINT "x es igual a 1"
  CONT
CASE 2
  PRINT "x es igual a 2"
CASE 1,3
  PRINT "x es igual a 3"
DEFAULT
  PRINT "x no es igual a 1,2 y 3"
ENDSELECT
```

-->En el monitor se visualiza 'x es igual a 1' y 'x es igual a 2'.

El comando CONT hace que se saltee la instrucción CASE o DEFAULT que figura detrás de él. En este ejemplo, el valor detrás de la primera instrucción CASE es igual a la condición de ramificación.
Por lo tanto, se ejecuta la instrucción PRINT "x es igual a 1". A

continuación sigue una instrucción CONT, que saltea la línea CASE 3, a pesar de que no se cumple la condición de ramificación, o sea que es diferente de uno. Esto lleva a que también se ejecute la instrucción PRINT "x es igual 2". Cuando se alcanza la siguiente instrucción CASE o DEFAULT se salta detrás de ENDSELECT. Esto también ocurre, cuando (como acá) detrás de este CASE se indica la expresión de la condición de bifurcación .

Ejemplo:

```
SELECT INP(2)
CASE "a" TO "g"
  PRINT "a hasta g"
DEFAULT
  PRINT "default"
ENDSELECT
```

--> Cuando pulsa una de las teclas a,b,c,d,e,f o g, se visualiza el texto 'a a g', cuando pulsa otra tecla (no teclas de conmutación del teclado) aparece 'default'.

En CASE se pueden combinar las diferentes posibilidades , o sea no es necesario escribir:

```
SELECT a$
CASE "a" TO "z"
CONT
CASE "A" TO "Z"
CONT
CASE "ae", "oe", "ue", "ss", "Ae", "Oe", "Ue"
PRINT "OK"
ENDSELECT
```

sino que también alcanza con escribir :

```
SELECT a$
CASE "a" TO "Z", "A" TO "Z", "ae", "oe", "ue",...
```

Estructuras de repetición o loops

FOR, TO, STEP, NEXT
REPEAT, UNTIL
WHILE, WEND
DO, LOOP, DO WHILE, DO UNTIL, LOOP WHILE, LOOP UNTIL
EXIT IF

El GFA-BASIC 3.0 dispone de una cantidad muy grande de tipos de estructuras de repetición o loops. Normalmente se distingue entre estructuras de repetición de rechazo y de no rechazo. Las estructuras de repeticón que revisan la condición de interrupción antes de la entrada al loop, se llaman de rechazo, mientras que aquellas que la revisan al final de la estructura de repetición se llaman de no rechazo. En consecuencia, los loops de no rechazo se deben correr al menos una vez.

El GFA-BASIC conoce en primer lugar a dos estructuras de no rechazo convencionales, el loop FOR-NEXT y el REPEAT-UNTIL. Un tercer tipo estándar de estructuras de repetición es el loop de rechazo WHILE-END. Como cuarto tipo de estructura de repetición se dispone de DO-LOOP. Este tipo en realidad solo es una estructura de repetición infinita

sin condición de interrupción, aunque en el GFA-BASIC se puede aplicar en forma muy variable. Tanto detrás de DO como detrás de LOOP pueden figurar las ampliaciones WHILE y UNTIL, de tal manera que se pueden construir estructuras de repetición, que al comienzo y al final del loop solicitan una condición lógica.

En cada una de estas estructuras de repetición se pueden incorporar además cualquier cantidad de condiciones de interrupción (EXIT IF).

```
FORi=aTOe[STEPs]
(instrucciones)
NEXTi
DOWNTO

i: avar
a,e,s: aexp
```

La estructura de repetición FOR-NEXT (también llamado loop contador) sirve para ejecutar repetidas veces un grupo de sentencias (cuerpo del loop) que figura entre las palabras FOR y NEXT. A la variable i, que se usa como contador, se le asigna primero el valor inicial a . Luego se revisan las sentencias del cuerpo del loop hasta que se alcanza NEXT. Allí se incrementa la variable i en una cantidad igual al valor que arroje s, que figura a continuación de la cláusula STEP. Si se omite la cláusula STEP en un loop FOR...NEXT, entonces se asumirá de que se trata de un salto unitario, o sea STEP=1. Ahora se analiza , si i superó el valor e. Si este es el caso se transfiere el control del programa a la sentencia siguiente al NEXT, caso contrario se vuelve a comenzar con la primera sentencia del cuerpo del loop. Este proceso se repite hasta que i supere a e.

Una consecuencia de esta ejecución es que después de abandonar la estructura de repetición i siempre es igual al primer valor que superó al criterio de interrupción. El contenido del loop FOR...NEXT se debe pasar al menos una vez.

En lugar del valor del STEP s=-1 se puede usar también la instrucción DOWNTO en vez de TO. STEP no se puede usar más en relación con DOWNTO.

Las variables que se usan como contador deberían ser variables enteras, ya que las sentencias del loop se ejecutan entonces con mayor velocidad que con variables de punto flotante. Esto naturalmente no es posible con steps fraccionarios.

En lugar de la palabra NEXT seguida de la variable usada como contador (por.ej.i%) se puede escribir también endfor i%, lo que el intérprete reemplaza por NEXT i%.

Ejemplos:

```
FOR i=1 TO 10
  PRINT i'
NEXT i
FOR i=-1 DOWNTO -10
  PRINT i'
NEXT i
```

--> En la pantalla se visualizan los números
1 2 3 4 5 6 7 8 9 10 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10

```
a$="P*r*u*e*b*a"
FOR j=1 TO LEN(a$) STEP 2
  PRINT MID$(a$,j,1);
NEXT j
```

--> En la pantalla aparece 'Prueba'.

```
REPEAT
(instrucciones)
UNTILbed

bed:  bexp
```

Con REPEAT - UNTIL bed se enmarca una cantidad de instrucciones, que se ejecutan hasta que la condición lógica bed es verdadera.

Cuando se alcanza la instrucción REPEAT en el programa, se ejecutan las sentencias que le siguen hasta que se alcanza UNTIL. Entonces se examina, si la condición lógica bed que figura detrás de UNTIL es verdadera. Si es este el caso, se ejecutan las sentencias que le siguen a UNTIL. Pero si bed da falso (0), la ejecución del programa salta nuevamente a la instrucción REPEAT.

Las instrucciones entre REPEAT y UNTIL se ejecutan por lo menos una vez, en la medida en que no se abandonan con EXIT o GOTO. Se trata de un loop de no rechazo.

En lugar de UNTIL bed también se puede escribir ENDREPEAT bed, lo que el intérprete reemplaza automáticamente por UNTIL.

Ejemplos:

```
REPEAT
UNTIL MOUSEK
```

--> Espera que se pulse una tecla del ratón.

```
i=1
REPEAT
  INC i
  j=SQR(i)
UNTIL i>10 and FRAC(j)=0
PRINT i
```

--> En la pantalla se visualiza el número 16.

```
WHILEbed
(instrucciones)
WEND

bed:  bexp
```

Las instrucciones WHILE y WEND pueden incluir un grupo de sentencias que se ejecutan mientras que la condición lógica bed sea verdadera. Cuando el GFA-BASIC se topa con la instrucción WHILE, se analiza la condición lógica que le sigue. Si es verdadera, se ejecutan las sentencias entre WHILE y WEND. Cuando se alcanza WEND, el programa

vuelve a saltar a WHILE y el ciclo vuelve a comenzar, hasta que bed sea falso. En lugar de la instrucción WEND también se puede escribir ENDWHILE, lo que el intérprete reemplaza por WEND.

Ejemplo:

```
WHILE INKEY$=""
  PLOT MOUSEX,MOUSEY
WEND
```

--> Dibujar con el ratón hasta que se pulse una tecla. Si ya figura un carácter en el buffer del teclado, no se fija un punto.

```
DO
(instrucciones)
LOOP
```

Las instrucciones DO LOOP producen una estructura de repetición infinita. El programa ejecuta las sentencias que figuran entre DO y LOOP y vuelve a saltar al comando Do cuando alcanza LOOP.

La estructura de repetición se puede abandonar únicamente con EXIT IF, GOTO o con instrucciones para finalizar programas. En lugar de la instrucción LOOP se puede escribir también ENDDO, lo que el intérprete reemplaza por LP.

Ejemplo:

```
DEFFILL 1,2,4
DO
  MOUSE mx,my,mk
  IF mk
    PBOX mx,my,mx+25,my+25
  ENDIF
LOOP
```

-->Cuando se pulsan teclas del ratón,dibuja rectángulos rellenos en la posición actual del cursor del ratón.

```
DO WHILEbed
DO UNTILbed
LOOP WHILEbed
LOOP UNTILbed

bed:  bexp
```

Las instrucciones DO y LOOP se pueden ampliar agregando UNTIL y WHILE. El encabezamiento del loop DO WHILE lleva a que el interior del loop se ejecute únicamente mientras que bed sea verdadero. Cuando el loop comienza con DO UNTIL, solo se accede cuando la condición bed no se cumple.

LOOP WHILE lleva a que el control del programa retorne a DO, mientras que bed es verdadero. Para LOOP UNTIL vale, que bed debe ser falso, para que se vuelva a saltar al comienzo del loop.

Por eso las condiciones son de rechazo en DO y de no rechazo en LOOP.

```
DO WHILE bed                        WHILE bed
.                                   .
.                 equivale a        .
.                                   .
LOOP                                WEND

DO                                  REPEAT
.                                   .
.                                   .
.                                   .
LOOP UNTIL bed                      UNTIL bed
```

Las variantes de la instrucciones DO, DO WHILE y DO UNTIL se pueden combinar de cualquier manera con LOOP, LOOPWHILE y LOOPUNTIL, de tal manera que con estas instrucciones se pueden construir en total nueve tipos de loops.

Ejemplos:

```
DO
LOOP UNTIL MOUSEK
```

--> Espera la pulsación de una tecla del ratón.

```
DO UNTIL MOUSEK=2
  DO WHILE MOUSEK=1
    LINE 0,0,MOUSEX,MOUSEY
  LOOP
LOOP UNTIL INKEY$="a"
```

--> Dibuja líneas cuando se mantiene oprimida la tecla izquierda del ratón.

```
DO UNTIL EOF(#1)
  INPUT #1,a$
LOOP
```

--> Lee secuencialmente cadenas de caracteres del canal 1, hasta que alcanza el final del archivo.

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

--> La construcción con WHILE WEND es más lenta, ya que además se necesita NOT.

```
EXIT IFbed

bed:  bexp
```

Con EXIT IF se puede saltar de una estructura de repetición, cuando se cumple la condición de Bool. El tipo de loop se puede elegir libremente en este caso.

Contrariamente a versiones anteriores de GFA-BASIC EXITIF también es posible dentro de IF-ENDIF y de SELECT-ENDSELECT.

Ejemplo:

```
DO
  EXIT IF MOUSEK
LOOP
REPEAT
  EXIT IF INKEY$="x"
UNTIL FALSE
```

--> Finaliza el programa, si primero se pulsa una tecla del ratón y luego la tecla 'x'.

Procedimientos y funciones

En el GFA-BASIC 3.0, igual que en cualquier lenguaje de programación moderno, los subprogramas llevan nombres. A estos subprogramas se pueden transferir parámetros, también se pueden aplicar sobre diferentes variables en distintos llamados a través de parámetros VAR.
Se puede transferir el valor de la variable como también la variable misma (VAR). En el segundo caso se puede modificar la variable de transferencia a través del procedimiento. O sea existe un "call by value" y un "call by reference".

También es posible el uso de variables locales, o sea no hay que tomar consideración de posibles colisiones de nombres con los procedimientos o funciones que llaman.

Con DEFFN se pueden definir funciones de una línea, que luego se activan por un nombre con FN. También son posibles funciones de varias líneas (FUNCTION). Estas solo representan una forma especial de los procedimientos y retornan un resultado (a través de RETURN).

```
GOSUBproc[(par1,par2,...)]
PROCEDUREproc[(var1,var2,...)]
RETURN
```

```
proc:  nombre del procedimiento
par1,par2: sexp, aexp
var1,var2:  svar, avar
```

Entre las palabras PROCEDURE y RETURN figuran las sentencias de un subprograma. Detrás de PROCEDURE figura el nombre del subprograma y eventualmente la lista de variables que se han de recibir. Un PROCEDURE se llama con la indicación de su nombre al comienzo de la línea, eventualmente seguido de los correspondientes parámetros que se fijan entre paréntesis. Para ilustrar que no se trata de instrucciones del GFA-BASIC, pueden ser precedidos del símbolo "@" o de la palabra clave "GOSUB". Esto es reglamentario cuando existen posibilidades de confusión con instrucciones del GFA-BASIC (por ej. @stop, @rem).

Los parámetros de transferencia pueden ser constantes, variables y expresiones. De variables se pueden transferir no solo los valores sino también las variables mismas (ver VAR).

Cuando durante la ejecución del programa se alcanza la instrucción RETURN, el programa salta a la sentencia que figura detrás de GOSUB. En lugar de la palabra PROCEDURE también se puede escribir SUB, en lugar de RETURN también ENDPROC o ENDSUB. Esto lo reemplaza el

intérprete por sí mismo.

Ejemplos:

```
GOSUB slow_print("** Manual de **")
@slow_print("* GFA-BASIC 3.0 *")
slow_print("GFA-SYSTEMTECHNIK")
'
PROCEDURE slow_print(t$)
  LOCAL i%
  FOR i%=1 TO LEN(t$)
    PRINT MID$(t$,i%,1);
    PAUSE 3
  NEXT i%
  PRINT
RETURN
```

--> La salida lenta, carácter por carácter, de un string.

```
a=8
@dritte_wurzel(a)
PRINT a
'
PROCEDURE dritte_wurzel(VAR x)
  x=x^(1/3)
RETURN
```

--> Calcula la raíz cúbica de 8 y en la pantalla se visualiza 2.

LOCALvar1,[,var2,var3,...]

var1,var2,var3: avar,svar

Con la instrucción LOCAL es posible limitar el rango de validez de variables. Las variables presentadas a continuación de LOCAL solo tienen validez dentro del procedimiento en el que figura LOCAL así como en todos los subprogramas llamados por este procedimiento.

Detrás de LOCAL pueden figurar variables, que tienen validez en el programa principal, o sea que tienen validez global. En la subrutina no se puede acceder a estas variables, pero están a disposición en forma inalterada cuando se abandona el procedimiento.

Las variables de transferencia de un procedimiento o de una función siempre son locales .

Ejemplo:

```
x=2
GOSUB test
PRINT x,y
'
PROCEDURE test
  LOCAL x,y
  x=3
  y=4
RETURN
```

--> En la pantalla se visualizan los números 2 y 0.

```
@funk[(par1,par2,...)]
FUNCTIONfunk[(var1,var2,...)]
RETURNexp
ENDFUNC
```

funk: nombre de la función
par1,par2: sexp, aexp
var1,var2: svar,avar
exp: sexp, aexp

Entre las palabras FUNCTION y ENDFUNC figuran las sentencias de una subrutina (similar a PROCEDURE). Detrás de FUNCTION figura el nombre de la subrutina y eventualmente la lista de las variables a transferir. Se accede a una subrutina indicando la arroba o signo at@ o FN y el nombre de la función. Luego sigue eventualmente una lista de parámetros.

Los parámetros de transferencia pueden ser constantes, variables y expresiones. De variables se pueden transferir no solo valores, sino también las variables mismas (ver VAR).

Cuando durante la ejecución del programa se alcanza la instrucción RETURN, se incorpora el valor calculado por la función en el lugar en que se llama a la misma.

En una función se puede aplicar varias veces RETURN , con IF o algo parecido, pero no está permitido finalizar una función sin RETURN (con ENDFUNC).

Un símbolo $ al final del nombre de una función caracteriza funciones con resultado string.

Ejemplo:

```
fl%=@fak_loop(15)
fr%=@fak_rekurs(10)
'
PRINT "Iteración:  fak(15) = ";fl%
PRINT "Recursión: fak(10) = ";fr%
'
FUNCTION fak_loop(f%)
  w=1
  FOR j%=1 TO f%
    MUL w,j%
  NEXT j%
  RETURN w
ENDFUNC
'
FUNCTION fak_rekurs(f%)
  IF f%<2
    RETURN 1
  ELSE
    RETURN f%*@fak_rekurs(PRED(f%))
  ENDIF
ENDFUNC
```

--> Las facultades de 10 y 15 se calculan dentro del loop y una vez en forma recursiva.

```
DEFFNfunc[(x1,x2,...)]=expr
FNfunc[(y1y2,...)]

func,x1,x2:  var
expr,y1,y2:  exp
```

La instrucción DEFFN permite definir funciones de una sola línea. La definición de esta función figura en la expresión expr la que puede rendir tanto una expresión numérica como un string. La función se llama después de su definición o bien con FN func o bien con la arroba o signo at@.

Las variables contenidas en la definición de la función se transfieren como parámetros. Para eso, en la definición de la función se anotan los nombres de las variables x1,x2,... que aparecen en la expresión expr. Los parámetros y1,y2,... se pueden transferir como expresiones numéricas o strings, cuando se llama la función.

Cuando las variables x1,x2,... también tienen definición global, no tienen acceso en la expresión expr, ya que dentro de expr contienen los valores transferidos. Por lo tanto, x1,x2,... son funciones para variables locales.

La instrucción FN se puede reemplazar también por @. Las funciones pueden estar encasilladas de cualquier manera, pero con DEFFN no en forma recursiva (la condición de interrupción no es posible).

Ejemplos:

```
DEFFN test(y,a$)=x-y+LEN(a$)
x=2
PRINT @test(4,"abcdef")
```

--> En la pantalla se visualiza el número 4(2-4+6).

```
DEFFN first_last$(a$)=LEFT$(a$)+RIGHT$(a$)
b$=@first_last$("TEST")
PRINT b$
```

--> En el monitor se visualiza el texto "TT".

```
DEFFN hoch_vier(x)=x^4
DEFFN vierte_wurzel(x)=x^(1/4)
PRINT @vierte_wurzel(@hoch_vier(1024))
```

--> Aparece el número 1024.

Ramificaciones relacionadas con un evento

En este apartado se tratan dos tipos de hechos, que se pueden solicitar en el GFA-BASIC. Los hechos no específicos del GFA-BASIC, como por ej. solicitar el clic del ratón, seleccionar menús pull-down, se analizan en otros capítulos (programación de menús y de ventanas, bibliotecas AES).

El primero de los dos hechos tratados es la pulsación simultánea de las teclas Control, Shift y Alternate. El segundo tipo de hecho es la aparición de un error.

```
ON BREAK
ON BREAK CONT
ON BREAK GOSUBproc
```

proc: nombre de un procedimiento

Estos tres comandos fijan la reacción a la pulsación simultánea de las teclas Control, Shift (solo la tecla Shift izquierda) y Alternate. La pulsación de esta combinación de teclas conduce normalmente a la interrupción del programa, pero también puede servir para saltar a un determinado procedimiento. Con este fin se fija el procedimiento proc con ON BREAK GOSUB.

Con ON BREAK GOSUB se logra que no se reaccione más a la pulsación de la combinación de teclas mencionada. ON BREAK vuelve a conectar la reacción normal (interrupción del programa).

Ejemplo:

```
ON BREAK GOSUB test
PRINT "Oprima CONTROL, SHIFT (izquierda) y ALTERNATE"
DO
LOOP
'
PROCEDURE test
  PRINT "Eso fue todo"
  ON BREAK
RETURN
```

--> Se solicita pulsar la combinación de teclas. Si esto ocurre, el procedimiento test vuelve a conectar la rutina break normal.

```
ON ERROR
ON ERROR GOSUBproc
RESUME[NEXT]
RESUME[mar]
```

proc: nombre de un procedimiento
mar: nombre de una marca

La ocurrencia de un error (errores TOS o específicos del GFA-BASIC) conduce normalmente a la impresión de un mensaje de error y a la interrupción del programa. Con ON ERROR GOSUB existe la posibilidad de ramificarse al procedimiento proc cuando ocurre un error. En este procedimiento se puede fijar la reacción frente a un error.

Con el comando ON ERROR se vuelve a conectar a la detección normal de errores, o sea a la impresión de un mensaje de error y a la detención inmediata del programa. Cuando ocurre un error, se ejecuta automáticamente un comando ON ERROR. Para poder reaccionar a varios errores sucesivos debe volver a aparecer el comando ON ERROR GOSUB proc en la rutina de detección de errores.

La instrucción RESUME permite reaccionar en forma especial cuando ocurre un error; solo tiene sentido en el procedimiento de detección de errores. Con RESUME NEXT se puede lograr saltar al siguiente comando que figura detrás de la sentencia del programa que condujo al error. Con RESUME mar se salta a la marca mar. La instrucción RESUME sin indicación de NEXT o de una marca vuelve a saltar a la sentencia

en la cual ocurrió el error. Cuando ocurrió un error fatal (ver FATAL) solo se puede usar la instrucción RESUME mar, pero no RESUME NEXT o RESUME sin indicación de la marca.

Ejemplo:

```
ON ERROR GOSUB fehlerabfanger
ERROR 5
PRINT "y otra vez ..."
ERROR 5
PRINT "no es alcanzado"
'
PROCEDURE fehlerabfanger
  PRINT "Ok, Error fue atajado."
  RESUME NEXT
RETURN
```

--> En la pantalla aparecen los textos 'ok, error detectado.' y ' y otra vez...'; luego el mensaje de error desencadenado por ERROR5 'raíz cuadrada solo para números positivos'. La marca para RESUME puede figurar tanto en un procedimiento como en el programa principal.

```
ERRORx
ERR
ERR$(x)
FATAL
```

x: aexp

Con ERROR se puede desencadenar el error con el número x (ver en el anexo la tabla de mensajes de error). Esto es útil por ej. cuando se revisa una rutina de manejo de errores.

En la var ERR figura el número del error ocurrido. Con ellas se pueden asignar reacciones específicas a determinados errores dentro de una rutina de manejo de errores.

La función ERR$ retorna el string del mensaje de error del GFA-BASIC con el número x.

La variable FATAL es verdadera, cuando el error ocurrido apareció en una dirección conocida para el intérprete. Esto puede ser por ej., cuando el error ocurrió en la ejecución de una rutina del sistema operativo. La consecuencia para el programa es que después de un error fatal no se puede ejecutar más en forma correcta un RESUME o RESUME NEXT.

Ejemplos:

```
ON ERROR GOSUB fehlerabfangung
INPUT "Elija el error: ",e
ERROR e
'
PROCEDURE fehlerabfangung
  PRINT "Este fue el error No.: ";ERR
  IF FATAL
    PRINT "que es un error fatal."
  ENDIF
RETURN
```

--> Solicita del usuario el número de error deseado y vuelve a visualizar este número.

```
FORM_ALERT(1,ERR$(100))
```

--> Se visualiza el mensaje de error con la caracterización 100 (o sea el mensaje Copyright) como Alert-Box.

Programación Interrupt

```
EVERYticksGOSUBproc
EVERY STOP
EVERY CONT
AFTERticksGOSUBproc
AFTER STOP
AFTER CONT
```

ticks: iexp
proc: nombre de un procedimiento

Con ayuda de las instrucciones EVERY y AFTER se puede acceder a procedimientos después de transcurrido un determinado tiempo ticks. Con la instrucción EVERY se accede al procedimiento proc en cada unidad de tiempo ticks; mientras que con AFTER se accede a este procedimiento una vez transcurridas ticks unidades de tiempo.

La unidad de tiempo se indica en dos centésimos de segundo (ticks=200 significa entonces un segundo). La ramificación al procedimiento indicado se puede ejecutar solo en cada cuarta unidad de tiempo, o sea que la presición efectiva del tiempo vale solo un cincuetavo de segundo.

Con EVERY STOP se puede detener el salto al procedimiento después de transcurrido el plazo de tiempo, y con EVERY CONT se vuelve a continuar. Las instrucciones AFTER STOP y AFTER CONT trabajan en forma análoga. Internamente en el GFA-BASIC se realizan estas instrucciones a través del vector del etv-timer ($400).

Recién después de la ejecución completa de una instrucción se analiza si se debe ejecutar un procedimiento de este tipo, o sea instrucciones que se ejecutan con lentitud como INP92), QSORT, operaciones de archivos o similares pueden impedir estas rutinas.

Ejemplo:

```
EVERY 4 GOSUB lines
linien!=TRUE
GRAPHMODE 3
DEFFILL 1,0
PLOT MOUSEX,MOUSEY
REPEAT
  IF MOUSEK=1
    EVERY STOP
  ELSE
    EVERY  CONT
  ENDIF
  DRAW TO MOUSEX,MOUSEY
UNTIL MOUSEK=2
'
```

```
PROCEDURE lines
  INC y%
  LINE 320,y%,639,y%
  IF y%=399
    y%=0
  ENDIF
RETURN
```

--> Produce el desplazamiento desde arriba para abajo de líneas en la mitad derecha del monitor y permite dibujar simultáneamente con el ratón. La pulsación de la tecla izquierda del ratón conecta y desconecta las líneas que se desplazan. El programa se puede detener pulsando la tecla derecha del ratón.

```
PRINT "En 3 segundos aparece un texto,"
PRINT "si no oprime ninguna tecla."
AFTER 600 GOSUB text
REPEAT
UNTIL INKEY$<>"" OR RAUS!
AFTER STOP
'
PROCEDURE text
  PRINT
  PRINT "Acá está el texto"
  raus!=TRUE
RETURN
```

--> Si en los tres segundos siguientes a la iniciación del programa no se pulsa una tecla, aparece un texto. Si se pulsa una tecla, entonces se finaliza el programa.

Otros casos

```
REM, GOTO, PAUSE, DELAY
END, EDIT
STOP
SYSTEM, QUIT
```

```
REMx
'x
<línea de sentencia>!x
```

x: cualquier texto

En una línea, que comienza con REM o con ', pueden figurar textos de cualquier tipo. Estos textos no están sometidos al control sintáctico del editor y no son atendidos en la ejecución del programa. Además, se puede agregar cualquier texto al final de la línea de sentencia, detrás del signo de exclamación !. El uso de este símbolo no es posible en relación con instrucciones DATA. También con INLINE no es posible el comentario.

Ejemplo:

```
REM Comentario
' PRINT "comentario"
PRINT "REM"  ! Comentarior
```

--> En la pantalla aparece la palabra REM.

```
GOTOmar
mar:
```

mar: marca definida por el programador

Con ayuda de una marca mar: se pueden fijar partes del programa, a las cuales se debe acceder con la instrucción GOTO. La ejecución del programa se continúa en la posición de la marca. La marca puede estar compuesta de letras, números, subrayado y puntos. También puede comenzar con un número, a diferencia de los nombres de variables, pero debe finalizar con 'dos puntos'. Cuando se accede a la marca con GOTO no se indican los 'dos puntos'.

Con GOTO no es posible entrar o salir de procedimientos o funciones. Esto también vale para estructuras de repetición FOR-NEXT. Con las instrucciones GOTO se pierde claridad del programa, por lo que , de ser posible, se deberían evitar.

Ejemplo:

```
PRINT "Posición 1"
GOTO sprungmarke
PRINT "Posición 2"
sprungmarke:
PRINT "Posición 3"
```

--> En la pantalla se visualizan los textos "Lugar 1" y "Lugar 3".

```
PAUSEx
DELAYx
```

x: aexp

La instrucción PAUSE contiene la ejecución del programa para x/50 segundos. DELAY tiene un efecto semejante, aunque la ejecución del programa se interrumpe por x segundos (presición teórica en milisegundos). DELAY usa la rutina GEM EVNT_TIMER, o sea se recomienda para programas GEM.

Ejemplo:

```
PRINT "Comienzo"
PAUSE 100
PRINT "una pausa"
DELAY 2
PRINT "Fin"
```

--> Aparecen el texto 'Start' y dos segundos más tarde 'una pausa', y después de otros 2 segundos 'Fin'.

```
END
EDIT
STOP
```

Estas instrucciones sirven para indicarle al intérprete de la máquina que el programa que se halla bajo ejecución ha finalizado. La instrucción END detiene la ejecución del programa y deja aparecer un

Alert-box con el texto 'Fin del programa'. El GFSA-BASIC 3.0 retorna al editor después de seleccionar la única alternativa de elección.

EDIT termina la ejecución del programa y retorna inmediatamente al editor.

STOP deja aparecer un Alert-box con las alternativas de elección STOP y CONT. Si se elige CONT se puede proseguir la ejecución del programa. Después de la elección de STOP el GFA-BASIC retorna al modo directo. Allí se pueden modificar o solicitar entonces los valores de variables y volver a proseguir la ejecución del programa con CONT.

Ejemplos:

```
x=3
STOP
PRINT x
```

--> Elija el Button 'STOP', cuando aparece el correspondiente Alert-box. Ingrese ahora las siguientes instrucciones en modo directo:

```
PRINT x
```

--> Aparece el número 3

```
x=4
CONT
```

--> Ahora se ejecuta la última sentencia del listado (PRINTx). Aparece el número 4, o sea el valor indicado en el modo directo.

```
NEW
```

Esta instrucción borra un programa. En el modo directo hay una pregunta de confirmación. En el editor se accede a esta instrucción con Shift-F4 o con el click del ratón (con pregunta de confirmación).

```
LOADf$
```

f$: sexp

La instrucción LOAD sirve para cargar un programa del GFA-BASIC. La expresión string f$ contiene el camino de acceso del archivo deseado. Si no se indica un sufijo, se preselecciona el uso de *.GFA .

Ejemplo:

```
LOAD "A:\TEST.GFA"
```

--> Carga un archivo del programa TEST.GFA del índice básico de la disquetera A.

```
SAVEf$
PSAVEf$
```

f$: sexp

La instrucción SAVE almacena un archivo de programa bajo el nombre indicado en f$. Cuando se usa la instrucción PSAVE se almacena el

archivo indicado con protección de listado (o sea el archivo no se puede listar cuando se vuelve a cargar con LOAD, sino que se ejecuta inmediatamente). Sin ninguna indicación se emplea *.GFA .

Ejemplo:

```
SAVE "A:\TEST.GFA"
```

--> Almacena el archivo de programa actual bajo el nombre TEST.GFA en la disquetera A.

```
LIST[f$]
LLIST[f$]
```

f$: sexp

La instrucción LIST presenta el programa actual en la pantalla. Opcionalmente se puede indicar un camino de acceso, bajo el cual se almacenó el archivo del programa en el formato ASCII. Archivos de programas, que se deben insertar en otros programas con MERGE, se deben almacenar con LIST o SAVE,A (de la zona del menú del editor) en formato ASCII.

Sin indicación de un sufijo se preselecciona el uso de .LST. El programa actual se puede imprimir en la impresora con LLIST. El listado de la impresora solo se puede interrumpir desconectando la impresora. Luego tarda aprox. otros 30 segundos hasta que el programa se sigue ejecutando o bien retorna al editor (comparar LLIST y los comandos de punto en el apartado sobre el editor). Pero también es posible desviar la impresión (ver más arriba).

Ejemplo:

```
LIST "A:\TEST.LST"
```

--> Guarda el programa actual bajo el nombre TEST.LST en formato ASCII sobre el disco en la disquetera A

```
.ll 70
.pl 66
LLIST
```

--> Imprime el programa actual en la impresora con una longitud de línea de 70 caracteres y 66 líneas por página.

```
CHAINf$
```

f$: sexp

La instrucción CHAIN posibilita la carga de un programa de GFA-BASIC en la memoria de trabajo e inicia la ejecución del el programa. Sin indicación de un sufijo se preselecciona el uso de *.GFA .

Ejemplo:

```
CHAIN "A:\ZUSATZ.GFA"
```

--> Carga el archivo del programa ZUSATZ.GFA e inicia el programa.

RUN[f$]

f$: sexp

La instrucción RUN inicia el programa actual. Si adicionalmente se indica el nombre completo de un archivo, entonces se carga e inicia el correspondiente programa.

Ejemplo:

```
RUN "A:\TEIL_2.GFA"
```

--> Carga e inicia el programa con el nombre PARTE_2.GFA de la disquetera A.

```
SYSTEM[n]
QUIT[n]
```

n: iexp

Estas instrucciones SYSTEM y QUIT tienen el mismo efecto. Finalizan la ejecución del programa y abandonan el GFA-BASIC. Contrariamente a las versiones 2.xx SYSTEM y QUIT retornan opcionalmente un valor entero de 2 bytes. Este Word es nulo, si se abandonó correctamente el intérprete y se transfiere al programa llamado (en el caso normal el Desktop).

En este caso vale la convención, que 0 señala una ejecución sin errores, un número positivo de 16 bits caracteriza un error interno o una advertencia y un número negativo de 16 bits representa generalmente un mensaje de error del sistema operativo. No todos los programas cumplen esto.

Ejemplos:

```
RESERVE 100
PRINT EXEC(0,"GFABASIC.PRG","","")
```

--> Iniciar este pequeño programa (Shift-F10), volver a cargar un GFA-BASIC. En este GFA-BASIC ingresar e iniciar la instrucción:

```
QUIT 23
```

--> Impresión: 23 y fin del programa del GFSA-BASIC que se cargó primero.

Manejo de errores

```
TRON
TRON#n
TROFF
```

n: iexp

La instrucción TRON (trace on) lleva a que en la pantalla se listen todas las sentencias ejecutadas. Esta lista se puede transferir a un archivo, a la impresora o a una interfase seriada etc. indicando el número de canal. La instrucción TROFF lo vuelve a desconectar.

Ejemplos:

```
PRINT "Comienzo:"
TRON
FOR i%=1 TO 5
  PRINT i%
NEXT i%
TROFF
PRINT "Fin"
```

--> Se visualiza la palabra 'Comienzo'. Luego aparecen los números del 1 a 5 y las instrucciones, que conducen a su representación, y finalmente la palabra 'Fin'.

```
OPEN "O",#1,"\tron.lst"
TRON #1
FOR i%=1 TO 10
  PRINT i%
NEXT i%
TROFF
CLOSE #1
'
OPEN "O",#2,"prn:"
TRON #2
FOR i%=10 TO 630 STEP 10
  LINE i%,0,i%,100
NEXT i%
TROFF
CLOSE #2
```

--> Se visualizan los números 1 a 10 y las instrucciones necesarias para ello y una serie de líneas verticales en intervalos de 10 pixel. La salida de estas instrucciones tiene lugar en el disco o en la impresora.

```
TRONproc
TRACE$
```

proc: nombre de un procedimiento

Con la instrucción TRON proc se puede indicar un procedimiento, que se puede llamar antes de tratar cada sentencia. La variable TRACE$ contiene la instrucción, que se ha de ejecutar como próxima. La instrucción TRON proc en combinación con TRACE$ hace posible una búsqueda de errores muy eficiente. Así, dependiendo por ej. de la siguiente instrucción a tratar , pueden salir determinadas variables en la impresora, de tal manera que se puede seguir la modificación de una variable durante la ejecución del programa.

Es importante, que el procedimiento TRON moleste lo menos posible. Lo mismo vale para las instrucciones PRINT en máscaras de la pantalla (TEXT, ATEXT,...). Tampoco se deberían usar rutinas de VDI (por uso de GDOS o DEFTEXT-LINE-A). O...

Ejemplo:

```
TRON tr_proc
GRAPHMODE 3
DO UNTIL MOUSEK
```

```
  x1%=100+RAND(200)
  y1%=100+RAND(100)
  x2%=200+RAND(200)
  y2%=200+RAND(100)
  PBOX x1%,y1%,x2%,y2%
LOOP
'
PROCEDURE tr_proc
  IF BIOS(11,-1) AND 4   ! Control-Taste
    adr%=XBIOS(2)
    BMOVE adr%+1280,adr%,4*1280
    PRINT AT(1,5);SPACE$(80);
    PRINT AT(1,5);LEFT$(TRACE$,79);
    PAUSE 20
  ENDIF
RETURN
```

--> Este programa dibuja rectángulos distribuídos al azar en la pantalla. Pulsando la tecla de Control se visualizan en la pantalla las sentencias recién tratadas. Pulsando una tecla del ratón se finaliza el programa.

```
DUMP[a$[TOb$]]
```

a$,b$: sexp

Con la instrucción DUMP se pueden visualizar los contenidos de variables durante la ejecución del programa o se pueden listar labels, procedimientos y funciones. Para ello la expresión a$ puede asumir los siguientes valores:

Ejemplos:

```
DUMP
```
--> Se visualizan todos los valores de variables y se dimensionan arreglos.

```
DUMP "a"
```
--> Igual que en el caso anterior, pero solo variables o arreglos que comienzan con 'a'.

```
DUMP ":"
```
--> Lista todos los labels (marcas) e indica luego el número de línea del editor, donde se definió la marca. La variación (:b) solo lista marcas, que comienzan con 'b'.

```
DUMP "@"
```
--> Lista todos los procedimientos y funciones, en cuyo caso figura detrás del nombre el número de línea, en la cual se encuentra éste en el editor:

```
proc_nombre @ 100        (procedimiento)
func_nombre FN 200       (función con retorno numérico)
func_nombre$ FN 300      (función con regtorno de string)
```
Labels, procedimientos y funciones que no se definen más se indican sin número de línea. Cuando el listado del programa se almacena bajo la opción SAVE,A y se vuelve a cargar, no vuelven a aparecer estos nombres indefinidos. Pero aparecen nombres de labels, procedimientos y funciones sin número de línea empleados pero no definidos.

A los números de línea indicados detrás de los nombres se puede acceder en el editor con Control+G.

Cuando se visualizan strings se representan como máximo 60 caracteres. Cuando la cadena de caracteres es más larga, se visualiza como último carácter un '>', sino un ' " '. Los caracteres de control del cursor se representan con '.'.

Las salidas mencionadas anteriormente se pueden transferir también a un archivo. Para ello se debe indicar un nombre de archivo en b$.

Como sufijo default o por defecto (sin.) se acepta .DMP.

8- Representación gráfica

El ATARI ST conoce tres modos diferentes de representación gráfica, un modo blanco y negro y dos modos color. Las coordenadas de las instrucciones gráficas y los colores que se pueden representar con ellas dependen de la resolución actual. Por lo tanto presentaremos un panorama de los modos de representación gráfica:

	Puntos de la pantalla (Coordenadas)	Colores (Registro de colores)
Resolución baja:	320 x 200 (0 a 319) (0 a 199)	16 (0 a 15) de 512 colores
Resolución media:	640 x 200 (0 a 639) (0 a 199)	4 (0 a 3) de 16 colores
Resolución alta:	640 x 400 (0 a 639) (0 a 399)	2 (0 a 1)

En el primer apartado se presentan las instrucciones para la selección de colores (SETCOLR, COLOR). Luego se indican las instrucciones de definición para la selección y producción de diferentes formas del cursor del ratón, marcas, diseños de relleno, encuadros y tipos de líneas (DEFMOUSE, DEFMARK, DEFFILL, BOUNDARY, DEFLINE).

El siguiente apartado presenta las diferentes instrucciones CLIP para la delimitación de representaciones gráficas en la pantalla y las instrucciones gráficas para dibujar diferentes formas geométricas básicas (PLOT, LINE, BOX, CIRCLE, ELLIPSE). Se describen posibilidades para la representación de polígonos (POLYLINE, POLYMARK, POLYFILL) y de textos gráficos (TEXT). El apartado concluye con la presentación de la instrucción FILL.

En el último apartado de este capítulo se discute el tratamiento de secciones de la pantalla con SGET, SPUT, GET y PUT.

Instrucciones de definición

```
SETCOLORregistro,rojo,verde,azul
SETCOLORregistro, valor mixto
COLORcolor
```

registro, rojo, verde, azul, valor mixto, color: iexp

La primer variante de SETCOLOR determina la proporción de los colores rojo, verde y azul en un determinado registro de colores. La intensidad de las proporciones de colores se fija en este caso con una escala de 0(escaso) a 7(grande). La cantidad de colores disponibles depende de la resolución actual. En la segunda variante se calcula el ajuste de color a través de la siguiente fórmula: valor mixto = rojo * 256 + verde * 16 + azul, donde los valores para rojo, verde y azul provienen nuevamente de la escala de 0 a 7.

Fijar la proporción de colores en los registros de color naturalmente solo tiene sentido en las resoluciones de color. Para la resolución monocromática vale, que cuando para el valor mixto se indica un valor

distinto de 0 o 1, los números pares tienen el mismo efecto que 0 y los números impares tienen el mismo efecto que 1.

La instrucción COLOR determina el color del dibujo. Delante de la expresión numérica color figura un valor entre 0 y 15 que depende de la correspondiente resolución.

Ejemplo:

```
SETCOLOR 0,0
```

--> Sobre el monitor monocromático aparece letra blanca sobre fondo negro.

```
DEFMOUSEsímbolo
DEFMOUSEbitmuster$

símbolo:  iexp
bitmuster:  sexp
```

La primera variante de instrucciones determina el símbolo del ratón de una de las 8 formas predefinidas. Vale la siguiente asignación:

```
0 --> flecha
1 --> doble paréntesis
2 --> abeja
3 --> mano señalando
4 --> mano abierta
5 --> Cruz reticular fina
6 --> Cruz reticular gruesa
7 --> Cruz reticular enmarcada
```

La segunda variante sirve para definir un cursor propio del ratón. Para eso se deben transferir el punto de acción, el color de la máscara, el color del cursor y el diseño de bits para fijar el aspecto de la máscara y el aspecto del cursor del ratón en una cadena de caracteres. El punto de acción es aquel punto del cursor del ratón, cuyas coordenadas se administran como posición del ratón. Cuando se solicita la posición del ratón, se retornan las coordenadas del punto de acción.

Se debe prestar atención que todos estos valores se transfieran en el tamaño de palabra. Para eso se puede usar la instrucción MKI$, de tal manera que bitmuster$ resulta así:

```
bitmuster$=MKI$( coordenada x del punto de acción)
+MKI$( coordenada y del punto de acción)
+MKI$(1)     !normal,  -1=XOR
+MKI$(color de la máscara)
+MKI$(color del cursor)
+máscara$ (diseño de bits de la máscara)
+cursor$ (diseño de bits del cursor)
```

Máscara$ y cursor$ consisten de 16 palabras cada uno, los cuales a su vez tienen el diseño de bits de una linea.

Ejemplo:

```
DEFMOUSE 2
REPEAT
UNTIL MOUSEK
m$=MKI$(0)+MKI$(0)+MKI$(1)+MKI$(0)+MKI$(1)
FOR i%=1 TO 16
  m$=m$+MKI$(65535)
NEXT i%
FOR i%=1 TO 16
  m$=m$+MKI$(1)
NEXT i%
PBOX 0,0,200,100
DEFMOUSE m$
REPEAT
UNTIL MOUSEK
```

--> Primero aparece una abeja como cursor del ratón. Después de pulsar una tecla del ratón, el cursor del ratón tiene la forma de una raya. Sobre el fondo negro en el extremo izquierdo superior aparece la forma de un rectángulo originado por la definición de la máscara. El loop de espera se abandona pulsando una tecla del ratón.

DEFMARK[color],[tipo],[tamaño]

color,tipo,tamaño: iexp

DEFMARK fija el color, tipo y tamaño de los cantos, que se fijan con la instrucción POLYMARK. Para la expresión numérica color existen los valores entre 0 y 15 según la resolución (ver el resumen de este capítulo). Con tipo se pueden hacer las siguientes marcas:

1 --> punto
2 --> signo más
3 --> asterisco
4 --> rectángulo
5 --> cruz
6 --> rombo

Valores mayores producen un asterisco como forma de marca. Los valores para el tamaño de la marca se indican en pixel. Si solo se quiere usar el segundo o tercer parámetro de la instrucción, se puede suprimir por ej. el primer parámetro y fijar únicamente las comas correspondientes para separar los parámetros. DFMARK ,4 significa por ej. que los dos primeros parámetros conservan su ajuste actual, pero el tamaño de las marcas se fija en 4.

Ejemplo:

```
DIMx%(1),y%(1)
x%(0)=50
y%(0)=50
x%(1)=150
y%(1)=150
DEFMARK 1,4,2
POLYMARK 2,x%(),y%()
DEFMARK ,3,4
POLYMARK 2,x%(),y%() OFFSET 30,0
```

--> Dibuja dos pares de puntos con diferentes marcas en los puntos finales.

```
DEFFILL[color],[estilo],[diseño]
DEFFILL[color],bitmuster$
```

color,estilo,diseño: iexp
bitmuster$: sexp

Esta instrucción fija el diseño de relleno para las instrucciones PBOX, PCIRCLE, PELLIPSE, POLYFILL y FILL. Determina el color, el estilo y el diseño de relleno y posibilita definir diseños creados por uno mismo. Para la variable color se pueden emplear valores entre 0 y 15 según la resolución (ver el resumen de este capítulo). Para estilo vale la siguiente asignación:

0 --> vacío
1 --> relleno
2 --> punteado
3 --> rayadfo
4 --> símbolo ATARI

Con diseño se pueden seleccionar diseños de 24 puntos o 12 líneas (ver anexo: tabla de diseños de relleno).

En la segunda variación de instrucciones se transfieren en bitmuster$ la información de bits de 32 bytes necesaria para la definición del diseño de relleno de 16*16 pixel. Estas informaciones deben presentarse en formato de palabra, para lo cual se ofrece la instrucción MKI$.

Los parámetros que figuran adelante se pueden suprimir, si se indican las comas de separación de parámetros. Así, por ej. DEFFILL ,2,4 selecciona el diseño de relleno ,2,4 y no modifica el color de relleno.

En la resolución media del sistema color puede haber después de las 16 palabras del primer bitplane otras 16 palabras para el segundo bitplane. En resolución baja son entonces en total 4 * 16= 64 palabras para diseños de relleno multicolor.

Ejemplos:

```
DEFFILL 1,2,4
PBOX 10,10,40,40
BOX 50,50,100,100
FILL 70,70
FOR i=1 TO 16
  f$=f$+MKI$(RAND(65535))
NEXT i
BOX 100,100,150,150
DEFFILL 1,f$
FILL 120,120
```

--> Dibuja un box en el diseño de relleno preseleccionado y uno en diseño al azar.

```
DO
  FOR j%=0 TO 15
    f$=""
    s%=BCHG(s%,j%)
    FOR i%=1 TO 16
      f$=f$+MKI$(s%)
    NEXT i%
    DEFFILL 1,f$
    PBOX 0,0,639,399
  NEXT j%
LOOP
```

--> Define un diseño de relleno y se visualiza un rectángulo con un diseño.

```
FOR i%=1 TO 64
  READ a%
  a$=a$+MKI$(a%)
NEXT i%
DEFFILL ,a$
PBOX 20,20,300,200
DATA -1,-1,-1,-1,-1,-1,-1,-1, 0, 0, 0, 0, 0, 0, 0, 0
DATA -1,-1,-1,-1, 0, 0, 0, 0,-1,-1,-1,-1, 0, 0, 0, 0
DATA -1,-1, 0, 0,-1,-1, 0, 0,-1,-1, 0, 0,-1,-1, 0, 0
DATA -1, 0,-1, 0,-1, 0,-1, 0,-1, 0,-1, 0,-1, 0,-1, 0
```

--> Esta rutina crea un diseño de relleno, que completa en el monitor blanco y negro dos gruesas franjas blanco y negro. Para el monitor color resultan con resolución media cuatro franjas de medio grosor en los cuatro colores posibles y con resolución baja 16 franjas de cada vez una línea de altura en todos los 16 colores.

BOUNDARYn

n: iexp

La instrucción BOUNDARY usa la función de VDI vsf_perimeter y conecta o desconecta automáticamente el enmarcado de la superficie de relleno. Cuando n es igual cero, se desconecta el enmarcado y cuando n es distinto de cero, se conecta.

Ejemplo:

```
DEFFILL 1,2,2
BOUNDARY 1              ! conectar el enmarcado
PBOX 50,50,100,100
BOUNDARY 0              ! desconectar el enmarcado
PBOX 150,50,200,100
```

--> Dibuja un rectángulo relleno con y sin enmarcado.

DEFLINE[estilo], [espesor],[comienzo_s,final_es]

estilo,espesor,comienzo_s,final_es: iexp

La instrucción DEFLINE determina el aspecto de las líneas, que se trazan con las instucciones LINE, BOX, RBOX, CIRCLE, ELLIPSE y POLYLINE. El primer parámetro estilo fija el estilo de las líneas.

Aquí se puede elegir entre diseños predefinidos y definidos por uno mismo. Son posibles los siguientes diseños predefinidos:

0 --> línea en el color de fondo
1 --> línea completa
2 --> línea de trazos de pequeños intervalos
3 --> línea punteada
4 --> línea de trazos y punteada
5 --> línea de trazos de gandes intervalos
6 --> el diseño de la línea resultante es: trazo-punto-punto...

Los diseños definidos por uno mismo se fijan mediante la indicación de un valor de 16 bit, donde un bit fijado equivale a un punto fijado (en el modo monocromático).

El segundo parámetro espesor determina el espesor de la línea en pixel. El espesor de la línea solo puede asumir valores impares. Cada espesor de línea indicado es tratado como el número que le sigue en orden decreciente.

Con comienzo_s y final_es se determina el símbolo de comienzo y final del tipo de línea. Acá se puede elegir entre:

0 --> angular
1 --> flecha
2 --> circular

Se pueden suprimir los parámetros que figuran al comienzo, si se presentan las comas de separación. Así, por ej. con DEFLINE "1,1 como símbolo para el comienzo y final de línea se fija la flecha, pero no se modifican el estilo y el ancho (ver tabla en el anexo).

Ejemplo:

```
FOR i=1 TO 6
  DEFLINE i
  LINE 50,i*50,200,i*50
NEXT i
DEFLINE 1,1,1,2
FOR i=2 TO 12 STEP 2
  DEFLINE ,i
  LINE 250,i*25,400,i*25
NEXT i
DEFLINE -&X1010101010101010,1,0,0
LINE 500,10,500,390
VOID INP(2)
```

--> Se dibujan líneas en los seis estilos predefinidos. Luego se generan líneas de distinto espesor. La línea en el diseño definido por uno mismo es punteada. Finalmente se espera a que uno pulse una tecla.

DEFTEXT[color],[attr],[ángulo],[altura],[fontnr]

color,attr,ángulo,altura,fontnr: iexp

Esta instrucción fija el aspecto que tendrá una cadena de caracteres que se imprimirá con TEXT.

Color tiene en este caso y según la resolución un valor entre 0 y 15.

Con attr se fijan los siguientes atributos del texto, los cuales se pueden combinar agregando las características:

```
0 --> normal      (normal)
1 --> bold        (grueso)
2 --> light       (fino)
3 --> italic      (inclinado)
4 --> underlined  (subrayado)
16--> outlined    (enmarcado)
```

La palabra ángulo determina la dirección de la impresión del texto. El valor se indica en 1/10 grados contra el sentido del reloj.

```
                    900
                     I
                     I
                     I
1800  ---  Punto cero del texto  --- 0
                     I
                     I
                     I
                    2700
```

Se permite el uso de los siguientes números:

```
0 --> Preseleccionado, de izquierda a derecha
900 --> de arriba para abajo
1800 --> de cabeza, de derecha a izquierda
2700 --> torsido de arriba para abajo
```

La palabra altura indica la altura de una letra mayúscula del texto en pixel. Sin embargo, en la puntuación normal solo se pueden leer sin dificultades las siguientes alturas de escritura:

```
4 --> altura de escritura Icon
6 --> letra pequeña
13 --> letra normal
32 --> letra especialmente gande
```

Finalmente se puede indicar con fontnr el número de una puntuación deseada, la cual previamente había sido instalada en órden (ver también: VST_LOAD_FONT,VQT_NAME,...)

Ejemplo:

```
FOR i|=0 TO 5
  DEFTEXT 1,2^i|,0,13
  TEXT 100,i|*16+100,"Atributo de texto "+STR$(i|)
NEXT i|
```

--> Se visualiza un texto de ejemplo con diferentes atributos.

```
GRAPHMODE n
```

n: iexp

La instrucción GRAPHMODE fija en qué forma se conectarán las salidas gráficas en la pantalla. La expresión numérica n figura en este caso

para los cuatro modos posibles:

```
1 --> replace                (reemplazar)
2 --> transparent            (transparente)
3 --> xor                    (invertir)
4 --> reverse transparent    (invertido y transparente)
```

Cuando n tiene el valor 1, o sea el modo preseleccionado, el nuevo diseño tapa al anterior. Con n igual 2 se conectan el diseño anterior y el nuevo mediante OR.

Cuando n es iguala 3 se fija cada pixel que anteriormente no se había fijado y se borra cada pixel que se había fijado anteriormente (o sea se invierten los diseños con la conexión XOR).

En el modo 4 se invierte primero el diseño nuevo y se conecta luego con OR.

Ejemplo:

```
FOR i%=1 TO 4
  GRAPHMODE i%
  DEFFILL 1,3,8
  PBOX 150*i%-100,10,150*i%,100
  DEFFILL 1,2,10
  PBOX 150*i%-140,50,150*i%-40,150
NEXT i%
```

--> El ejemplo muestra 4 rectángulos con diseños diferentes, los cuales están conectados entre sí en los cuatro modos.

Instrucciones generales para la representación gráfica

En este apartado se indican en primer lugar las posibilidades para delimitar impresiones de gráficos con las variantes de la instrucción CLIP.

Las instrucciones generales para la representación gráfica PLOT, LINE, BOX, RBOX, CIRCLE y ELLIPSE dibujan puntos, líneas, rectángulos, rectángulos con ángulos redondeados, círculos y elipses. Con PBOX,PRBOX, PCIRCLE y PELLIPSE se los puede rellenar con colores y diseños.

POLYLINE dibuja un esquema de polígono, a cuyos ángulos se pueden asignar diferentes símbolos con POLYMARK. POLYFILL rellena este polígono con un diseño en determinado color. Point da el color de determinado punto de la pantalla. FILL rellena una determinada zona enmarcada del monitor. Con TEXT se visualizan cadenas de caracteres en determinadas zonas de la pantalla. CLS borra la pantalla. El origen de las coordenadas para instrucciones de representaciones gráficas se encuentra en el extremo izquierdo superior. Las coordenadas pueden tener valores fuera de la resolución del monitor, aunque se representan solo las zonas visibles de la pantalla. Al final del capítulo se presenta la instrucción BITBLT.

```
CLIPx,y,w,h[OFFSETx0,y0]
CLIPx1,y1TOx2,y2[OFFSETx0,y0]
CLIP#n[OFFSETx0,y0]
CLIP OFFSETx,y
CLIP OFF

x,y,w,h,x0,y0,x1,y1,x2,y2,n:  iexp
```

Este grupo de instrucciones se usan para el 'clipping'. Quiere decir, para delimitar las salidas de representaciones gráficas de una sección rectangular de la pantalla. CLIP fija el 'clipping' para instrucciones de representaciones gráficas de VDI mientras que ACLIP lo hace solo para rutinas de gráfica de LINE-A. Para fijar esta sección de la pantalla (clipping rectangle) se deben trasmitir las coordenadas de los cantos enfrentados en diagonal o bien el ancho y la altura del rectángulo de limitación.

En este caso se distinguen varias variantes de la instrucción CLIP. CLIP x,y,w,h hace posible indicar la coordenada x izquierda 'x' y la coordenada 'y' superior así como el ancho 'w' y la altura 'h' del rectángulo de limitación.

Otra posibilidad consiste en usar la instrucción CLIP x1,y1 TO x2,y2. Aquí se transfieren las coordenadas de los cantos enfrentados en diagonal (x1,y1) y (x2,y2).

La tercera variante hace posible delimitar la impresión al tamaño de la ventana con el número 'n'.

En las variantes mencionadas anteriormente es posible fijar el origen de las impresiones gráficas en el punto con las coordenadas x0,y0, si se agrega OFFSET x0,y0 a la instrucción. La instrucción OFFSET x0,y0 también puede figurar sola, satisfaciéndose el mismo objetivo y fijándose el origen para las impresiones gráficas en el punto (x0,y0). El 'clipping' se desconecta con la instrucción CLIP OFF. La delimitación de las impresiones gráficas no vale para las instrucciones GET, PUT, BITBLT ni tampoco para todos los llamados de LINE-A e instrucciones AES.

```
PLOTx,y
LINEx1,y1,x2,y2
DRAW[TO][x,y]
DRAW[x1,y1][TOx2,y2][TOx3,y3][TO...]

x,y,x1,y1,x2,y2:  iexp
```

Plot dibuja un punto con las coordenadas x,y en la pantalla. LINE dibuja una línea entre los pares de coordenadas x1,y1 y x2,y2. Estilo y color de estas líneas se pueden fijar con DEFLINE y COLOR.

DRAW x,y equivale a la instrucción PLOT. Con DRAW TO x,y se traza una línea entre las coordenadas x,y y el último punto fijado. Es indiferente, si este punto se fija con PLOT, LINE o DRAW. Otra variante de la instrucción DRAW está representada por DRAW x1,y1 TO x2,y2. Esto equivale a la instrucción LINE. Además se pueden agregar otras indicaciones de coordenadas, con las cuales se pueden generar por ej. polígonos. La última variante de la instrucción DRAW posibilita la indicación de comandos, que son similares a determinados comandos de representaciones gráficas de LOGO (gráfica de Turtle) y

del lenguaje estándar Plotter HPGL de Hewlett-Packard. Así es posible simular un plotter en la pantalla.

Ejemplos:

```
x=50
y=50
farbe=POINT(x,2*y)
PRINT farbe
PLOT x,2*50
LINE 200,200,400,100
PRINT POINT(x,100)
```

--> Dibuja un punto y provee su color de dibujo.

```
DO
  MOUSE mx,my,mk
  IF mk=1
    DRAW TO mx,my
  ENDIF
  EXIT IF mk=2
LOOP
```

--> Cuando se pulsa la tecla izquierda del ratón se dibuja una comunicación entre las coordenadas actuales del ratón mx,my y el último punto dibujado. El loop se abandona presionando la tecla derecha del ratón.

```
DRAWexpresión
DRAW(i)
SETDRAW
```

```
i: iexp
expresión: una secuencia de sexp y aexp, el primero debe ser
sexp, la separación debe ser con comas, semicolon
comillas simples.
```

Con DRAW se mueve un lápiz imaginario a lo largo de la pantalla y en cierta forma se puede dibujar. Esto es parecido a las instrucciones de representaciones gráficas del lenguaje de programación Logo. Los parámetros de las sentencias son cada vez números de punto flotante, que también se pueden indicar en strings. Existen los siguientes comandos:

FD n	ForwarD	Desplaza el 'lápiz'n pixel 'hacia adelante'
BK n	BacKward	Desplaza el 'lápiz' n pixel 'hacia atrás'
SX x	Scale X	Fija la escala del 'movimiento del lápiz' para FD y BK con el factor indicado.
SY y	Scale y	La fijación de escala con SX y SY solo actúa sobre las instrucciones FD y BK. Con SX0 o bien SY0 se desconecta la escala (más rápido que con la fijación de escala con factor 1 (SX1, SY1)).
LT w	Left Turn	Indica el ángulo 'w', según el cual gira la dirección del dibujo hacia la izquierda.
RT w	Right Turn	En forma análoga pero hacia la derecha.
TT w	Turn To	Indica el ángulo absoluto 'w', para lo cual vale la siguiente asignación:

```
            0
            I
            I
270  ---  Cero  ---  90
            I
            I
           180
```

Las datos de w=ángulo se realizan en grados.

MA x,y	Move Absolute	Desplaza el 'lápiz' a las coordenadas absolutas x e y
DA x,y	Draw Absolute	Desplaza el 'lápiz' a las coordenadas absolutas x e y y traza una línea en el color actual de la última posición del punto (x,y).
MR xr,yr	Move Relative	Como 'MA', pero con respecto a la última posición.
DR xr,yr	Draw Relative	Como 'DR' pero en relación con la última posición.

Una abreviatura de la expresión DRAW "MA",x,y,"TT",w está dada por la instrucción SETDRAW x,y,w.

CO c	Color	Fija el color 'c' como color de 'dibujo'. (Comparar también parámetros en la instrucción COLOR).
PU	Pen Up	Levanta el 'lápiz'.
PD	Pen Down	Baja el 'lápiz'.

Adicionalmente se dispone de las siguientes funciones de consulta:

```
DRAW(0)   da   la posición x (como valor de punto flotante)
DRAW(1)   da   la posición y (como valor de punto flotante)
DRAW(2)   da   el ángulo en grados (como valor de punto flotante)
DRAW(3)   da   la escala del eje x (como valor de punto flotante)
DRAW(4)   da   la escala del eje y (como valor de punto flotante)
DRAW(5)   da   penflag (-1=PD, 0=PU)
```

Ejemplos

```
DRAW "ma 160,200 tt0"     ! trazar a partir de 160,200 con ángulo 0
FOR i&=3 TO 10
  eck(i&,90)              ! dibujar un polígono con i vértices
NEXT i&
'
PROCEDURE eck(n&,r&)      ! n=Número de vértices, r=largo del canto
  LOCAL i&
  FOR i&=1 TO n&
    DRAW "fd",r&,"rt",360/n&
  NEXT i&
RETURN
```

--> Dibuja polígonos.

```
FOR i=0 TO 359 STEP 8
  SETDRAW 320,200,i
  GRAPHMODE 3
  DRAW "fd 45 rt 90 fd 45 rt 90 fd 45 rt 90 fd 45"
  DRAW "bk 90 rt 90 bk 90 rt 90 bk 90 rt 90 bk 90"
  GRAPHMODE 1
  DRAW "fd 45 rt 90 fd 45 rt 90 fd 45 rt 90 fd 45"
  DRAW "bk 90 rt 90 bk 90 rt 90 bk 90 rt 90 bk 90"
NEXT i
```

--> Dibuja un rectángulo pequeño y uno grande, que giran sobre su propio eje.

```
l%=48
' raute 1
DRAW "ma60,100 tt45"
DRAW "fd",l%," rt90 fd",l%," rt90 fd",l%," rt90 fd",l%," rt90"
' rhomboid, schmal
DRAW "mr100,0 tt45"
DRAW "sx0.5 sy0"
DRAW "fd",l%," rt90 fd",l%," rt90 fd",l%," rt90 fd",l%," rt90"
' raute, breit
DRAW "mr100,0 tt45"
DRAW "sx0 sy0.5"
DRAW "fd",l%," rt90 fd",l%," rt90 fd",l%," rt90 fd",l%," rt90"
' raute, breit und hoch
DRAW "mr100,0 tt45"
DRAW "sx3 sy2"
DRAW "fd",l%," rt90 fd",l%," rt90 fd",l%," rt90 fd",l%," rt90"
```

--> Dibuja tres rombos y un romboide, para ello se usa como objeto de partida un rombo y las demás figuras se obtienen modificando la escala x e y.

```
BOXx1,y1,x2,y2
PBOXx1,y1,x2,y2
RBOXx1,y1,x2,y2
PRBOXx1,y1,x2,y2

x1,y1,x2,y2:  iexp
```

BOX dibuja un rectángulo con las coordenadas de los cantos x1,y1 y x2,y2 enfrentadas en diagonal. En forma análoga, PBOX dibuja un rectángulo relleno, RBOX un rectángulo de cantos redondeados y PRBOX un rectángulo relleno con cantos redondeados.

Ejemplo:

```
BOX 20,20,120,120
RBOX 170,20,270,120
x=150
DEFFILL 1,2,4
PBOX 20,20+x,120,120+x
PRBOX 170,20+x,270,120+x
~INP(2)
```

--> Dibuja un rectángulo, luego un rectángulo de cantos redondeados y finalmente nuevamente ambas formas rellenas. A continuación espera que se pulse una tecla.

```
CIRCLEx,y,r[,w1,w2]
PCIRCLEEx,y,r[,w1,w2]
ELLIPSEx,y,rx,ry[,w1,w2]

x,y,r,w1,w2:  iexp
```

CIRCLE dibuja un círculo cuyo centro tiene las coordenadas x,y y cuyo radio es r. Además se pueden indicar el ángulo inicial w1 y final w2 para determinar el arco circular. Análogamente PCIRCLE dibuja un círculo relleno o un segmento circular relleno.

ELLIPSE dibuja una elipse con las coordenadas de centro x,y y el radio horizontal rx y el radio vertical ry. Además se pueden indicar el ángulo inicial w1 y final w2 para determinar un arco de elipse. Análogamente PELLIPSE dibuja una elipse rellena o un segmento de elipse relleno. En este caso se indican los ángulos en 1/10 grados (0-3600) y los arcos y segmentos circulares se dibujan en sentido contrario al reloj (o sea 0 grados hacia la derecha, 90 grados hacia arriba, 180 grados hacia arriba y 270 grados hacia abajo).

Ejemplo:

```
CIRCLE 320,200,100
ELLIPSE 320,200,200,100,900,1800
PCIRCLE 320,200,100,1800,2700
PELLIPSE 320,200,200,100,2700,3600

--> Dibuja algunos círculos y (partes de) elipses.
```

```
POLYLINEn,x(),y()[OFFSETx_off,y_off)
POLYMARKn,x(),y()[OFFSETx_off,y_off]
POLYFILLn,x(),y()[OFFSETx_off,y_off]

n,x_off,y_off:  iexp
x(),y():  arreglo de avar
```

POLYLINE traza una línea con n cantos. Las coordenadas x,y de los cantos figuran en los campos x() e y(). El primer canto figura en x(0) e y(0), el último en x(n-1) e y(n-1). El primer y último canto se conectan automáticamente entre sí. Además se puede agregar a estas coordenadas un OFFSET horizontal (x_off) o vertical (y_off).

POLYFILL rellena un cuadrado con un diseño y un color fijados con DEFFILL.

Con POLYMARK se pueden marcar los puntos.

Ejemplo:

```
DIM x%(3),y%(3)
FOR i%=0 TO 3
  READ x%(i%),y%(i%)
NEXT i%
DATA 120,120,170,170,70,170,120,120
POLYLINE 4,x%(),y%()
POLYFILL 3,x%(),y%() OFFSET -50,-50
DEFMARK ,4,10
POLYMARK 3,x%(),y%() OFFSET 40,-80
~INP(2)
```

--> Dibuja un triángulo relleno y no relleno así como marcas de rectángulos de otro triángulo.

POINT(x,y)

x,y: iexp

Se obtiene el color del punto con las coordenadas x,y y se retorna a la variable. En la resolución baja se retornan valores entre 0 y 15, en la resolución media entre 0 y 3 y en la resolución alta solo entre 0 y 1.

Ejemplo:

```
a=POINT(100,100)
PLOT 100,100
PRINT a,POINT(100,100)
```

--> Escribe en la pantalla el color en la coordenada del monitor 100,100 antes y después de la instrucción PLOT.

FILLx,y[,f]

x,y,f: iexp

Esta instrucción rellena una superficie de cualquier forma. El proceso de relleno comienza en las coordenadas x,y. Cuando se indica f se delimita el proceso de relleno a través de puntos del color f y el márgen del monitor.

Si no se indica f, o con f=-1, se usa cada punto con otro color que el punto de partida (x,y) como límite.

Ejemplos:

```
LINE 0,180,639,180
FOR i=1 TO 19
  BOX i*20,100,i*20+20-i,180
  TEXT i*20-4,195,i
  DEFFILL ,2,i
  FILL i*20+1,101
NEXT i
```

--> Dibuja una recta, sobre la cual se alinean rectángulos con diferentes diseños y los rellena sin límite de color, lo cual lleva a la destrucción del diseño.

```
LINE 0,280,639,280
FOR i=1 TO 19
  BOX i*20,200,i*20+20-i,280
  TEXT i*20-4,295,i
  DEFFILL ,2,i
  FILL i*20+1,201,1
NEXT i
```

--> Dibuja debajo de ésto una recta con rectángulos sucesivos de diferentes diseños y los rellena con delimitación de color. Así se conservan los diferentes diseños de color.

```
CLS[#n]

n:  iexp
```

Borra la pantalla dejando visualizar ESC-E-CR. También se puede transferir a archivos.

Ejemplo:

```
PBOX 100,100,500,200
REPEAT
UNTIL MOUSEK
CLS
```

--> Rellena la pantalla parcialmente con rectángulos y la borra si se pulsa una tecla del ratón.

```
TEXTx,y[,l],expresión

x,y,l:  iexp
expresión:  sexp o aexp
```

En el punto de las coordenadas x,y imprime la expresión como texto del gráfico. Este punto se refiere al extremo izquierdo de la línea básica del primer carácter de la expresión. El parámetro l fija la longitud de la impresión del texto en pixel. Esto ocurre a través de la modificación de las distancia entre caracteres (con l>0) o a través de la modificación de la distancia entre palabras (con l<0). Cuando l=0, la expresión sale en forma inalterada en la pantalla.

El texto del gráfico puede ser provisto de atributos con DEFTEXT. Sin embargo, DEFTEXT actúa solo sobre la instrucción TEXT y la instrucción PRINT dentro de una ventana.

Ejemplo:

```
s$="Esto es un ejemplo."
FOR i=0 TO 23
  DEFTEXT 1,i,0,6
  TEXT 50,i*16+16,s$
NEXT i
DEFTEXT 1,0,0,13
TEXT 350,50,350-50
TEXT 350,100,s$
TEXT 350,150,250,s$
TEXT 350,200,-250,s$
~INP(2)
```

--> Escribe textos de gráficos en diferentes variantes sobre la pantalla y espera luego la pulsación de una tecla.

```
SPRITEbitmuster$[,x,y]

bitmuster$:  svar
```

La instrucción SPRITE hace posible desplazar por la pantalla un objeto de 16 pixel por 16 pixel de tamaño. Las correspondientes informaciones de bit para el diseño y la máscara se fijan en un string. Se debe prestar atención de transferir todos estos valores en el tamaño de

palabra. Con este fin se puede usar la instrucción MKI$, de tal manera que bitmuster resulta así:

```
bitmuster$=MKI$(coordenada x del punto de acción)
+MKI$(coordeneda y del punto de acción)
+MKI$(0) para normal o MKI$(-1) para XOR
+MKI$(Color de la máscara) generalmente 0
+MKI$(Color del sprite) generalmente 1
+sprite$
```

Sprite$ contiene las informaciones de bit para la forma y máscara del sprite, las cuales se deben indicar alternadamente, a diferencia de DEFMOUSE donde se indican una tras otra.

Ejemplo:

```
gfa$=MKI$(1)+MKI$(1)+MKI$(0)
gfa$=gfa$+MKI$(0)+MKI$(1)
FOR i%=1 TO 16
  READ muster%,maske%
  gfa$=gfa$+MKI$(maske%)+MKI$(muster%)
NEXT i%
DATA 0,0,0,32256,15360,16896,8192,24064
DATA 8192,24560,11744,21008,9472,23280,9472,23295
DATA 15838,16929,274,32493,274,749,286,737
DATA 18,1005,18,45,18,45,0,63
REPEAT
  ADD mx%,(MOUSEX-mx%)/50
  ADD my%,(MOUSEY-my%)/50
  SPRITE gfa$,mx%,my%
UNTIL MOUSEK=2
```

--> Desplaza un sprite a lo largo de la pantalla, el cual sigue lentamente al cursor del ratón.

Secciones de la pantalla

```
SGETpantalla$
SPUTpantalla$

pantalla$:  svar
```

SGET copia muy rápido toda la pantalla (32000 byte) en un string. SPUT copia en forma análoga un string de 32000 bytes de longitud en la memoria de la pantalla.

Ejemplo:

```
PCIRCLE 100,100,50
SGET b$
~INP(2)
CLS
~INP(2)
SPUT b$
```

--> Dibuja un círculo relleno sobre la pantalla. Después de pulsar una tecla desaparece y vuelve a aparecer cuando se pulsa nuevamente una tecla.

```
GETx1,y1,x2,y2,sección$
PUTx1,y1,sección$[,modo]

x1,y1,x2,y2,modo:  iexp
sección$:  svar
```

GET deposita una sección de la pantalla en una variable string. PUT coloca en forma análoga una sección de la pantalla depositada con GET en el punto x,y de la memoria de la pantalla. Con modo se puede determinar opcionalmente cómo se ha de conectar el diseño de bits depositado en sección$ con el contenido existente de la pantalla. En la siguiente tabla significa Q(origen) el diseño de bits contenido en el string y Z(meta) el contenido original de la pantalla.

Modo	Regla de conexión	Efecto
0	0	Se borran todos los puntos.
1	Q and Z	Solo quedan fijados aquellos puntos que están fijos en ambas matrices.
2	Q AND (NOT Z)	Solo se fijan los puntos que están fijados en la matriz de origen y borrados en la matriz de meta.
3	Q	La matriz de origen se transfiere en forma inalterada (preselección de GRAPHMODE 1).
4	(NOT Q) AND Z	Solo se fijan los puntos que están borrados en la matriz de origen y fijados en la matriz de meta.
5	Z	La meta no se altera.
6	Q FOR Z	Solo se fijan los puntos que están exactamente en una matriz (GRAPHMODE 3).
7	Q FOR Z	Se fijan todos los puntos que están fijados en una de las dos matrices (GRAPHMODE 2).
8	NOT (Q OR Z)	Se fijan todos los puntos que no están fijados en ninguna de las dos matrices.
9	NOT (Q XOR Z)	Se fijan todos los puntos, que están fijados en ambas matrices o en ninguna matriz.
10	NOT Z	La matriz de meta se invierte.
11	Q OR (NOT B)	Se fijan todos los puntos, que están fijados en la matriz de origen y todos los que no están fijados en la matriz de meta.
12	NOT B	La matriz de origen se invierte antes de la transferencia.
13	(NOT Q) OR B	GRAPHMODE 4
14	NOT (Q AND Z)	Son fijados todos los puntos, que no estaban contenidos en ambas matrices.
15	1	Se fijan todos los puntos.

Si se fija además el bit 4 del modo, entonces se produce adicionalmente una conexión AND del diseño de relleno con la matriz de origen.

VSYNC

Esta instrucción sirve para sincronizar la conformación de la pantalla

(o sea, el programa espera el 'vertical blank interrupt').

VSYNC se puede usar por ej. para animar partes de la pantalla con GET y PUT.

Ejemplo:

```
t%=TIMER
FOR i%=1 TO 100
  VSYNC
NEXT i%
PRINT SUB(TIMER,t%)/200
```

--> Se visualiza el tiempo necesario para 100 cambios de pantalla.

```
BITBLTs_mfdb%(),d_mfdb%(),par%()
```

s_mfdb%(),d_mfdb%(),par%(): arreglos de enteros

La instrucción BITBLT sirve para copiar secciones rectangulares de la pantalla. Tiene cierta similitud con las instrucciones GET y PUT aunque es más flexible y veloz, pero más complicada en la aplicación.

Los parámetros de la instrucción se depositan en tres campos. El primero describe la estructura del sector a copiar (matriz de origen) y el segundo el sector en que se depositará la copia (matriz de meta). El tercero contiene las coordenadas del sector de origen y de meta así como el modo de copia.

Esta instrucción se basa en una rutina VDI, mientras que BITBLTdir% y BITBLTx%() usan una rutina de LINE-A (ver apartado sobre llamados de LINE-A).

La estructura de la matriz de origen (s_mfdb%) y de meta (d_mfdb%) son iguales. Las abreviaturas significan:

s_mfdb%() source memory form description block
d_mfdb%() destination memory form description block

Los elementos del arreglo son:

_mfdb%(0) Dirección inicial de la matriz. Esta dirección debe ser par.
_mfdb%(1) Ancho de la matriz en pixel. Este valor debe ser divisible por 16.
_mfdb%(2) Altura de la matriz en pixel.
_mfdb%(3) Ancho de la matriz en palabras (=cantidad de pixel/16)
_mfdb%(4) Reservado, siempre 0.
_mfdb%(5) Cantidad de planos de las matrices:
Resolución alta = 1
Resolución media = 2
Resolución baja = 4

El campo par%() tiene la siguiente estructura:

par%(0) coordenada x izquierda de la matriz de origen
par%(1) coordenada y superior de la matriz de origen
par%(2) coordenada x derecha de la matriz de origen
par%(3) coordenada y inferior de la matriz de origen

```
par%(4)   coordenada x izquierda de la matriz de destino
par%(5)   coordenada y superior de la matriz de destino
par%(6)   coordenada x derecha de la matriz de destino
par%(7)   coordenada y inferior de la matriz de destino
par%(8)   modo de copia.
```

Los valores para el modo de copia equivalen a aquellos de GET/PUT (ver en el capítulo de Representación gráfica). Los principales son:

```
3 = replace              (GRAPHMODE 1)
6 = xor                  (GRAPHMODE 2)
7 = transparent          (GRAPHMODE 3)
13= invers transparent   (GRAPHMODE 4)
```

Ejemplo:

```
DIM smfdb%(8),dmfdb%(8),p%(8)
'
FOR i%=0 TO 639 STEP 8
  LINE i%,0,639,399
NEXT i%
'
GET 0,0,639,399,a$
mirrorput(0,0,a$)
'
PROCEDURE mirrorput(x%,y%,VAR x$)
  IF LEN(x$)>6      !solo si existe algo
    a%=V:x$
    b%=INT{a%}
    h%=INT{a%+2}
    '
    smfdb%(0)=a%+6
    smfdb%(1)=(b%+16) AND &HFFF0
    smfdb%(2)=h%+1
    smfdb%(3)=smfdb%(1)/16
    smfdb%(5)=DPEEK(a%+4)
    dmfdb%(0)=XBIOS(3)
    '
    dmfdb%(1)=640
    dmfdb%(2)=400
    dmfdb%(3)=40
    dmfdb%(5)=1
    '
    p%(1)=0
    p%(3)=h%
    p%(5)=y%
    p%(7)=y%+h%
    p%(8)=3
    p%(4)=x%+b%
    p%(6)=x%+b%
    '
    FOR i%=0 TO b%
      p%(0)=i%
      p%(2)=i%
      BITBLT smfdb%(),dmfdb%(),p%()
      DEC p%(4)
      DEC p%(6)
    NEXT i%
    '
```

```
  ENDIF
RETURN
```

--> Dibuja un gran círculo, en el cual se encuentra un círculo pequeño. Luego se lee toda la pantalla en un string y a continuación se refleja sobre el eje vertical (comparar BITBLT en el apartado llamados de LINE-A).

9 - Administración de eventos, menús y ventanas

Administración de eventos

Existen instrucciones específicas del GFA-BASIC, con las cuales se pueden solicitar eventos (en la literatura para GEM llamados events) en forma sencilla. Estos eventos son la pulsación de una tecla del ratón, la pulsación de una tecla, la supervisión de las coordenadas del ratón en relación con dos sectores rectangulares de la pantalla y la entrada de un "GEM-message", en el cual se encuentran por ej. informaciones par la administración de ventanas.

La supervisión de estos eventos se conecta mediante un ON MENU xxx GOSUB, donde xxx es el evento, al cual se debe reaccionar. La supervisión tiene lugar a través de la instrucción ON MENU. Cada vez que se llama esta instrucción se revisa, si ha tenido lugar un evento. Cuando apareció un evento frente al cual se ha de reaccionar, se bifurca hacia el procedimiento fijado a tal fin.

ON MENU[t]

t: iexp

La instrucción ON MENU supervisa la aparición de eventos. Antes de esta instrucción se debería fijar, a qué procedimiento debería bifurcarse el programa ante un determinado evento. Las instrucciones, con las que se fija ésto se describen en el resto de este capítulo. Para una constante supervisión de un evento es necesario ejecutar repetidas veces esta instrucción. Por eso, la instrucción del ON MENU se encuentra normalmente en una estructura de repetición.

El parámetro t contiene el tiempo (en milésimas de segundo), después del cual ha de finalizar la instrucción ON MENU. El objetivo de esta indicación del tiempo consiste en que el GEM de vez en cuando no nota que se deja de pulsar una tecla del ratón (efecto típico: se invierte el campo de cierre de una ventana, pero la ventana permanece abierta, hasta que se vuelve a pulsar enérgicamente varias veces más), lo cual no debería volver a ocurrir en el empleo de esta variante de instrucción. Sin embargo, la condición es supervisar también las teclas del ratón. Para ello se necesita una instrucción ON MENU BUTTON x,y,z GOSUB.

Ejemplo:

```
ON MENU BUTTON 1,1,1 GOSUB test
t%=TIMER
REPEAT
  PRINT (TIMER-t%)/200
  ON MENU 2000
UNTIL MOUSEK=2
PROCEDURE test
RETURN
```

--> Se visualiza cada dos segundos el tiempo desde el comienzo del programa. Si se pulsa la tecla izquierda del ratón, aparece el evento a descubrir, y ON MENU finaliza antes de transcurrido el lapso de dos segundos. La pulsación de la tecla derecha del ratón produce la detención del programa. Si se modifica la primera línea: ON MENU BUTTON 0,0,0 GOSUB test, se desconecta la supervisión del ratón, y el

parámetro del tiempo detrás de ON MENU no tiene más efecto.

MENU(x)

x: aexp entre -2 y 15 inclusive

En las variables MENU(-2) hasta MENU(15) se encuentran todas las informaciones relevantes para el manejo de eventos. Para el caso, en que se eligió una inscripción de menú en un menú pull-down, figura en MENU(0) el índice de la inscripción elegida en el arreglo de las inscripciones de menús (ver el siguiente apartado y programa de ejemplo al final del apartado sobre menús pull-down).

El MENU(-2) contiene la dirección del message-buffer, y en MENU(-1) se encuentra la dirección del cuerpo del menú pull-down.

En las variables MENU(1) a MENU(8) se encuentra el message-buffer y en MENU(9) a MENU(15) el bloque AES-Integer-Output. Ahora se discutirá brevemente la utilidad de las informaciones que se encuentran en estos lugares.

El punto de partida de este análisis es el MENU(1) y el message-buffer. En MENU(1) figura el número característico del evento aparecido. Los demás elementos del message-buffer contienen diferentes informaciones según el MENU(1). La siguiente tabla resume estas informaciones. Se indica en cada caso el valor de MENU(1), luego el significado de este valor y finalmente las variables, en las cuales se encuentran respectivas informaciones importantes a este valor del MENU(1). Estas informaciones son importantes especialmente para la correcta administración de ventanas.

MENU(1) = 10	Se seleccionó una inscripción de menú pull-down.
MENU(0)	Indice de la inscripción del menú en el arreglo de inscripción.
MENU(4)	Indice-objeto del título del menú.
MENU(5)	Indice-objeto de la inscripción de menú seleccionada.
MENU(1) = 20	Se debe volver a dibujar un fragmento rectangular de la ventana
MENU(4)	Manejo (handle) de la ventana.
MENU(5)	Coordenada x izquierda del fragmento de la ventana.
MENU(6)	Coordenada y superior del fragmento.
MENU(7)	Ancho del fragmento.
MENU(8)	Altura del fragmento (Ejemplo en ON MENU MESSAGE GOSUB).
MENU(1) = 21	Se activó una ventana (normalmente quiere decir que el usuario quiere activar esta ventana).
MENU(4)	Manejo (handle) de la ventana activada.
MENU(1) = 22	Se activó el campo de cierre de una ventana.
MENU(4)	Manejo (handle) de esta ventana.
MENU(1) = 23	Se activó el campo superior derecho de una ventana (normalmente quiere decir que el usuario quiere llevar la ventana a su máximo tamaño)
MENU(4)	Manejo (handle) de esta ventana.

MENU(1) = 24	Se activó uno de los cuatro campos de flechas o una zona de fijación del marco de la ventana. El desplazamiento de un regulador lleva a MENU(1)=25 o bien 26. Aquí solo se analiza, si se ha activado a la derecha o a la izquierda de la posición actual del regulador en la zona de fijación.
MENU(4)	Manejo (handle) de esta ventana.
MENU(5)	Qué ha sido activado? Vale la siguiente asignación: 0: por encima del regulador derecho 1: por debajo del regulador derecho 2: flecha hacia arriba 3: flecha hacia abajo 4: a la izquierda del regulador inferior 5: a la derecha del regulador inferior 6: flecha hacia la izquierda 7: flecha hacia la derecha
MENU(1) = 25	El regulador inferior ha sido desplazado.
MENU(4)	Manejo (handle) de la ventana.
MENU(5)	Posición del regulador (número entre 1 y 1000).
MENU(1) = 26	El regulador derecho ha sido desplazado.
MENU(4)	Manejo (handle) de la ventana.
MENU(5)	Posición del regulador (número entre 1 y 1000).
MENU(1) = 27	Se modificó el tamaño de la ventana con ayuda del elemento de manejo derecho inferior. El nuevo tamaño de la ventana se retorna en:
MENU(4)	Manejo (handle) de la ventana.
MENU(5)	Coordenada x izquierda.
MENU(6)	Coordenada y superior.
MENU(7)	Ancho de la ventana.
MENU(8)	Altura de la ventana.
MENU(1) = 28	Se modificó la posición de una ventana, las nuevas coordenadas figuran en:
MENU(4)	Manejo (handle) de la ventana.
MENU(5)	Coordenada x izquierda.
MENU(6)	Coordenada y superior.
MENU(7)	Ancho de la ventana.
MENU(8)	Altura de la ventana.
MENU(1) = 29	Se activó una nueva ventana con el GEM. Esto puede ocurrir al cerrarse una ventana ubicada arriba de ésta, si corre por ej. un accesorio.
MENU(4)	Manejo (handle) de la ventana.
MENU(1) = 40	Fue elegido un accesorio. Este valor solo puede ser recibido por un accesorio, que en base al valor que figura en MENU(4) ha de analizar si ha sido llamado o no.
MENU(4)	Número de identificación del menú del accesorio.
MENU(1) = 41	Se cerró un accesorio. Este valor solo puede ser recibido por un accesorio, el cual debe analizar en base al valor que figura en MENU(4) si ha sido finalizado o no.
MENU(4)	Número de identificación del menú del accesorio.

Las variables MENU(9) a MENU(15) y GINTOUT(0) a GINTOUT(7) contienen las siguientes informaciones, si en MENU(9) se fijó el bit para el correspondiente evento:

En MENU(9) figura un bit característico que describe el tipo de evento que ha ocurrido. La asignación es:

```
Bit 0 -->  teclado
Bit 1 -->  tecla del ratón
Bit 2 -->  ratón ha abandonado/ingresado al rectángulo 1
Bit 3 -->  ratón ha abandonado/ingresado al rectángulo 2
Bit 4 -->  ha tenido lugar un mensaje del message-buffer
Bit 5 -->  indicador del tiempo (timer)
```

```
MENU(10)   Posición x del ratón cuando se desencadena el evento.
MENU(11)   Posición y del ratón cuando se desencadena el evento
MENU(12)   Estado de las teclas del ratón, valiendo lo siguiente:
            0 -->  Ninguna tecla pulsada.
            1 -->  Tecla izquierda pulsada.
            2 -->  Tecla derecha pulsada.
            3 -->  Ambas teclas pulsadas.
                   (Ejemplo en ON MENU BUTTON x,y,z GOSUB).

MENU(13)   Estado de las teclas de conmutación; para cada tecla de
           conmutación pulsada, se fija un bit. Vale la siguiente
           asignación:
            Bit 0 --> Tecla Shift derecha
            Bit 1 --> Tecla Shift izquierda
            Bit 2 --> Tecla Control
            Bit 3 --> Tecla Alternate
                      (Ejemplo en ON MENU KEY GOSUB)
MENU(14)   Información a través de la tecla pulsada. En el byte in-
           ferior figura el código ASCII de la tecla pulsada, en el
           byte superior el código Scan (ejemplo en ON MENU KEY
           GOSUB).
MENU(15)   Cantidad de clics del ratón (activaciones), que condujeron
           al evento.
```

```
ON MENU BUTTON clics,tecla,estadoGOSUBproc
```

```
 clics,tecla,estado:  iexp
 proc: nombre de un procedimiento
```

Esta instrucción sirve para supervisar las teclas del ratón. En las variables clics,tecla y estado se define, qué estado de las teclas debe llevar a la ramificación hacia el procedimiento proc.

En clics figura la cantidad máxima de clics de las teclas del ratón a registrar. En tecla figura, qué combinación de teclas se espera. En este caso vale:

```
0 --> una tecla.
1 --> tecla izquierda.
2 --> tecla derecha.
3 --> ambas teclas simultáneamente.
```

La expresión estado indica, qué estado deben tener las correspondientes teclas. En este caso dice 0 para una tecla no pulsada y 1 para una tecla pulsada. El nombre proc indica, a qué procedimiento

se debe ramificar, cuando ha tenido lugar el evento descripto a través de clics,tecla y estado.

La supervisión tiene lugar con la instrucción ON MENU.

Ejemplo

```
ON MENU BUTTON 1,1,0 GOSUB box
GRAPHMODE 3
REPEAT
  ON MENU
UNTIL MOUSEK=2
'
PROCEDURE box
  ADD i%,7
  IF i%>200
    i%=3
  ENDIF
  BOX 320-i%,200-i%,320+i%,200+i%
RETURN
```

--> Dibuja un juego gráfico en la pantalla, mientras no se pulse la tecla izquierda del ratón. La pulsación de la tecla derecha del ratón finaliza el programa.

```
ON MENU KEY GOSUBproc
```

proc: nombre de un procedimiento

Esta instrucción instala una supervisión del teclado. proc es el nombre de un procedimiento, al cual se ha de ramificar, cuando se pulsa una tecla durante la instrucción ON MENU.

Ejemplo:

```
ON MENU KEY GOSUB key_auswertung
REPEAT
  ON MENU
UNTIL MOUSEK=2
'
PROCEDURE key_auswertung
  PRINT "Teclas de conmutación: ";MENU(13)
  PRINT "Código ASCII: ";BYTE(MENU(14))
  PRINT "Código Scan: ";SHR(MENU(14),8)
  PRINT
RETURN
```

--> Cuando se pulsa una tecla aparece el mensaje del estado actual de las teclas de conmutación del teclado (Shift,Control,Alternate) y los códigos ASCII y Scan de la tecla pulsada. La pulsación de la tecla derecha del ratón finaliza el programa. En MENU(x) ver los significados de MENU(13) y MENU(14).

```
ON MENU IBOXnr,x,y,b,hGOSUBproc
ON MENU OBOXnr,x,y,b,hGOSUBproc
```

nr,x,y,b,h: iexp
proc: nombre de un procedimiento

Estas dos instrucciones supervisan las coordenadas del ratón. Cuando el ratón entra en una zona rectangular de la pantalla (IBOX) o cuando la abandona (OBOX), se ramifica hacia el procedimiento proc.

Es posible en este caso, definir dos zonas rectangulares de la pantalla, que son supervisadas independientemente entre sí. Nr es el número del correspondiente rectángulo, x es su coordenada x izquierda, y es la coordenada y superior, b es el ancho y h la altura del rectángulo.

La supervisión tiene lugar durante la ejecución de una instrucción ON MENU.

Ejemplo:

```
ON MENU IBOX 1,250,130,140,140 GOSUB rein_in_box
ON MENU OBOX 2,50,50,540,300 GOSUB raus_aus_box
'
GRAPHMODE 3
BOX 250,130,390,270
BOX 50,50,590,350
REPEAT
  ON MENU
UNTIL MOUSEK=2
'
PROCEDURE rein_in_box
  BOX 250+i%,130+i%,390-i%,270-i%
  IF i%=70
    i%=2
  ENDIF
  ADD i%,2
RETURN
'
PROCEDURE raus_aus_box
  BOX 0+j%,0+j%,639-j%,399-j%
  IF j%=0
    j%=50
  ENDIF
  SUB j%,2
RETURN
```

--> Cuando el ratón entra en el rectángulo interno, tiene lugar allí un juego gráfico. Cuando abandona el rectángulo externo, el juego tiene lugar en el rectángulo externo. La pulsación de la tecla derecha del ratón finaliza el programa.

ON MENU MESSAGE GOSUBproc

proc: nombre de un procedimiento

Cuando entra un mensaje en el message-buffer, el programa se bifurca hacia el procedimeinto con el nombre proc. La supervisión del message-buffer tiene lugar en la instrucción ON-MENU. La estructura del message-buffer se discute en el apartado sobre el MENU(x).

Ejemplo:

```
DIM m$(10)
FOR i%=0 TO 10
  READ m$(i%)
NEXT i%
DATA Desk, Redraw ,-------------------------
DATA 1,2,3,4,5,6,"",""
OPENW 4,0,0
MENU m$()
ON MENU MESSAGE GOSUB message_auswertung
PRINT AT(1,1);
REPEAT
  ON MENU
UNTIL MOUSEK=2
'
PROCEDURE message_auswertung
  IF MENU(1)=20
    PRINT CHR$(7)
    PRINT "Un recorte de pantalla"
    PRINT "debe ser dibujado de nuevo."
  ELSE
    PRINT CHR$(7)
    PRINT "Ha pasado algo !!!"
  ENDIF
RETURN
```

--> Para probar este listado debe estar cargado un accesorio. Cuando se lo selecciona, el programa se da cuenta que ha sido tapado un fragmento de la pantalla y reporta el mensaje de ésto (ver también MENU(x)). Al comienzo del programa también aparece este mensaje una vez. El programa se puede finalizar pulsando la tecla derecha del ratón.

Menús pull-down

En este apartado se introducen las instrucciones específicas del GFA-BASIC 3.0 para la administración de menús pull-down. Lamentablemente existe en la literatura una cierta confusión de términos relacionados con este tema. Por eso se debe decir aquí en primer lugar, qué significado han de tener en este manual los términos relacionados con este tema. El término menú pull-down lo usaremos como concepto principal. En la línea superior de la pantalla se encuentra la parte permanentemente visible del menú pull-down, la zona del menú. Esta contiene los diferentes títulos. Cuando la flecha del ratón llega a uno de estos títulos, se "desdobla" debajo de este título un llamado menú. Cada parte de este menú, que se puede seleccionar individualmente, se llama inscripción de menú. Esta elección de términos no tiene carácter obligatorio general, sino que vale únicamente para este manual; en algunos libros se usa otra terminología.

Cuando se instala un menú pull-down, se fijan sus inscripciones en un arreglo string m$(). Este menú pull-down se lleva a la pantalla con la instrucción MENUm$(). La instrucción ON MENU GOSUB fija un procedimiento, al cual se puede ramificar después de la elección de una inscripción del menú. Durante la ejecución del programa se examina cada vez que aparece una instrucción ON MENU, si se ha seleccionado una inscripción.

La inscripción MENU OFF vuelve a representar inscripciones invertidas de la zona del menú en la letra normal. Con MENU KILL se desconecta el menú pull-down.

La instrucción MENUx,y permite tildar inscripciones de menú o representarlas en letra clara, lo que lleva a que esta inscripción de menú no se pueda seleccionar más .

```
ON MENU GOSUBproc
MENUm$()
```

```
 oric:  nombre de un procedimiento
 m$(): arreglo string
```

Estas dos instrucciones son responsables para la generación y administración de un menú pull-down, siendo apoyadas por las instrucciones y variables del apartado anterior (ON MENU, MENU()). Con ON MENU GOSUB proc se fija un procedimiento, hacia el cual se ha de ramificar después de la selección de una inscripción del menú. Si esta inscripción es un accesorio, no se activa este procedimiento. Dentro del procedimiento se puede detectar con la variable MENU(0), qué inscripción del menú ha sido elegida. MENU(0) es el índice de la inscripción elegida en el arreglo de las inscripciones m$(); m$(MENU(0)) es por lo tanto el texto activado en el menú pull-down, cuando vale OPTION BASE 0. Cuando se ha conectado OPTION BASE 1, el texto de la inscripción elegida es m$(MENU(0)+1).

MENUm$() representa el menú pull-down en la pantalla. En el campo de cadenas de caracteres m$() figuran los títulos, inscripciones y reservas de lugar para los accesorios. Se debe respetar el siguiente formato cuando se ordenan las inscripciones en el campo m$:

m$(0)	Título del menú, en el que puede haber accesorios.
m$(1)	Nombre de la primera inscripción en el primer menú.
m$(2)	Una secuencia de signos menos (-).
m$(3)-m$(8)	Reservas de lugar para los accesorios. Se trata únicamente de seis strings con un largo mayor que cero. Si se han cargado accesorios cuando se conectó el regulador, entonces aparecen en el menú pull-down bajo su nombre, caso contrario no figuran las correspondientes inscripciones de menú.
m$(9)	Un string nulo, que marca el final del primer menú.

Todos los demás menús tienen el siguiente formato:

1. Título del menú.
2. Lista de inscripciones del menú.
3. Un string nulo, que ,marca el final del menú.

Detrás del último menú sigue otro string nulo, que marca el final de todo el menú pull-down. Una inscripción de menú, que comienza con un signo menos, se representa automáticamente en forma clara y no se puede elegir.

Ejemplo: Ver final de este apartado

```
MENU OFF
MENU KILL
```

Cuando se elige una inscripción de menú se invierte el título del menú. MENU OFF revierte el título al estado normal.

MENU KILL desconecta el menú pull-down; aunque no se aleja de la pantalla. MENU KILL desconecta además las conexiones ON MENU GOSUB.

```
MENU x,y
```

x,y: aexp

Con esta instrucción se puede reubicar la n-sima inscripción de un menú pull-down. La enumeración de las inscripciones equivale a los índices del arreglo string, en el cual figuran las inscripciones del menú pull-down. El recuento comienza entonces en cero; los títulos, las inscripciones dummy para los accesorios y los strings nulos se cuentan.

El segundo parámetro y indica, qué se debe ocurrir con la x-sima inscripción. Aquí vale la siguiente asignación:

y	Efecto
0	--> sin tilde adelante.
1	--> fijar adelante un tilde
2	--> letra clara.
3	--> letra normal.

Ejemplo: Ver al final del apartado.

Programa de ejemplo para el apartado sobre menús pull-down

```
DIM eintrag$(20)
i%=-1
REPEAT
  INC i%
  READ eintrag$(i%)
UNTIL eintrag$(i%)="Final"
eintrag$(i%)=""
'
MENU eintrag$()
ON MENU GOSUB auswertung
OPENW 0
'
DATA Desk, Test ,--------------------,1,2,3,4,5,6,
DATA File, Load , Save ,------, Quit ,
DATA Título, Valor 1 , Valor 2 ,
DATA Final
'
REPEAT
  ON MENU
UNTIL MOUSEK=2
'
PROCEDURE auswertung
  MENU OFF
  ' MENU(0) contiene el Array-Index del item de menú
```

```
  m%=MENU(0)
  PRINT eintrag$(m%)
  '
  ALERT 0,"Quiere un tilde ?",1," Si | No ",a%
  IF a%=1
    MENU m%,1
  ELSE
    MENU m%,0
  ENDIF
  '
  ALERT 0,"Letra clara ?|(no seleccionable)",1," Si | No ",a%
  IF a%=1
    MENU m%,2
  ELSE
    MENU m%,3
  ENDIF
RETURN
```

--> Genera y administra un menú pull-down. Cuando se activa una inscripción del menú se escribe su texto en el monitor y se pregunta, cómo se ha de representar la inscripción del menú elegido.

Instrucciones para ventanas

El GFA-BASIC brinda algunas instrucciones sencillas para la administración de ventanas, las cuales no son muy flexibles (OPENW, CLOSEW, CLEARW, TITLEW, INFOW). Si se quiere programar una administración de ventanas eficiente, se deben usar o bien las correspondientes rutinas AES o fijar los parámetros de la ventana deseada en una tabla (WINDTAB) y volver a llamar la ventana. Además se dispone de las funciones W_HAND y W_INDEX como eslabón de unión entre instrucciones sencillas del GFA-BASIC para la administración de ventanas y las funciones AES amplias provenientes de la biblioteca de ventanas.

Las instrucciones sencillas parten de posiciones de ventanas poco variables. Un canto de la ventana siempre se encuentra en la misma posición, mientras que el otro canto se encuentra en el punto de contacto de las cuatro ventanas.

Cuando se usan las instrucciones sencillas se fija el punto cero del sistema de coordenadas para impresiones de texto y de gráficos en el extremo superior izquierdo de la ventana. Aunque esto no lo tienen en cuenta algunas instrucciones como por ej. GET y PUT, BITBLT,etc.. Las impresiones de gráficos y de textos que abandonan la ventana se cortan automáticamente (clipping).

```
OPENW#n,x,y,w,h,attr
OPENWnr[,x_pos,y_pos]
CLOSEW[#]nr

 nr,x_pos,Y_pos:  aexp
 n,x,y,h,attr:  iexp
```

Con OPENW se abre la ventana con el número nr. Los parámetros x_pos e y_pos determinan la posición de una canto de la ventana; en cada caso el canto enfrentado en diagonal es fijo . Para una administración de ventanas más variable se necesitan las rutinas AES o WINDTAB. Las

coordenadas (x,y) de las posibles ventanas son:

Nr	Parte izquierda superior	Parte derecha inferior
1	(0,19)	(x_pos,y_pos)
2	(x_pos,19)	(639,y_pos)
3	(0,y_pos)	(x_pos,399)
4	(x_pos,y_pos)	(639,399)

Por lo tanto, el punto (x_pos,y_pos) es el punto de contacto de las cuatro ventanas.

Con la instrucción OPENW 0 no se abre una ventana real, sino que se desplaza condicionalmente el origen de las coordenadas al punto (0,19). Entonces no se pueden alcanzar más las 19 líneas superiores de la pantalla con instrucciones de impresión de gráficos o de textos. Esto es útil, para no escribir sobre una zona de menú pull-down .

La instrucción CLOSEW cierra la ventana con el número nr, CLOSEW# cierra la ventana con el índice n.

Ejemplo: (ver también el programa ejemplo al final del apartado).

```
REPEAT
  IF MOUSEK=1
    CLOSEW 1
    OPENW 4,320,200
  ENDIF
  IF MOUSEK=2
    CLOSEW 4
    OPENW 1,100,100
  ENDIF
UNTIL MOUSEK=3
CLOSEW #1
CLOSEW #4
```

--> La pulsación de la tecla izquierda del ratón abre la ventana 11, la pulsación de la tecla derecha del ratón la ventana 4. Si se pulsan ambas teclas simultáneamente se finaliza el programa.

```
TITLEW #1,"Titel"                          ! öffnet Fenster mit Index 1
INFOW # 1,STRING$(15,"....|")              ! Infozeile vorbelegen
OPENW #1,16,32,600,300,&X111111111111      ! Koordinaten + Attribute setzen
~INP(2)
CLOSEW #1                                  ! wichtig!!, Fenster schlieen
```

--> Abre una ventana con línea de título e Info. La pulsación de una tecla finaliza el programa.

```
W_HAND(#n)
W_INDEX(#hd)
```

n, hd: aexp

W_HAND da el GEM-Handle de la ventana, cuya caracterización se indica en n. W_INDEX es la función inversa de la otra y retorna el número de ventana correspondiente al GEM-handle.

Ejemplo:

```
OPENW 2
PRINT W_HAND(#2)
~INP(2)
CLOSEW #2
```

--> Escribe el handle de la ventana con la caracterización 2 en la pantalla. Si se pulsa una tecla finaliza el programa.

```
CLEARW[#]nr
TITLEW[#]nr,a$
INFOW[#]nr,a$
TOPW#nr
FULLW[#]nr
```

La instrucción CLEARW borra la ventana con el número nr. TITLEW escribe en la línea superior de la ventana nr el texto a$. INFOW escribe en la línea de información subyacente de la ventana nr el texto a$. TOPW activa la ventana con el número nr. FULLW lleva la ventana nr a su total tamaño de pantalla.

Ejemplo:

```
DEFFILL 1,2,4
PBOX 0,0,639,399
OPENW 1
PAUSE 50
FULLW #1
PRINT "Ventana 1"
OPENW 4,100,100
PAUSE 50
CLEARW 1
OPENW 3
PAUSE 50
TOPW #1
PAUSE 50
CLOSEW #1
TITLEW 4,"Ventana 4"
INFOW 3,"Ventana 3"
PAUSE 100
CLOSEW #3
CLOSEW 4
```

--> Deja visualizar algunas ventanas, las modifica y las vuelve a descomponer.

```
WINDTAB
WINDTAB(i,j)
```

i,j: iexp

A partir de la dirección indicada en WINDTAB figura la dirección de la tabla de parámetros de la ventana, en la cual se encuentran todas las informaciones que determinan el aspecto de la ventana. En esta tabla se puede fijar además el punto cero de las impresiones de gráficos.

La tabla está compuesta de 68 bytes y se construye palabra por palabra

(2 bytes son cada vez una inscripción en el tabla). El uso de la tabla se ilustra al final del apartado en un programa de ejemplo. En este ejemplo se inscriben en la tabla de la ventana los parámetros de la ventana a construir y se abre luego la ventana con OPENW.

WINDTAB (similar a INTIN()) también se puede activar como campo bidimensional WINDTAB(). El primer índice es el número de la ventana (1 a 4 o 0), el segundo índice es:

0 para el handle, 1 para los atributos, 2 para la coordenada x, 3 para la y, 4 para el ancho y 5 para la altura de la ventana (medidas externas).

WINDTAB se puede activar también a través de DPOKE, los offsets individuales tienen el siguiente significado:

Offset	Función
0	Manejo (handle) de la ventana 1.
2	Atributos para la ventana 1 (Estructura ver más adelante).
4	Coordenada x para la ventana 1.
6	Coordenada y para la ventana 1.
8	Ancho para la ventana 1.
10	Altura para la ventana 1.
12 a 22	Los valores correspondientes para la ventana 2.
24 a 34	Los valores correspondientes para la ventana 3.
36 a 46	Los valores correspondientes para la ventana 4.
48	-1
50	0
52 a 58	Coordenadas y medidas para la ventana DESKTOP(0)
60 a 62	Coordenadas para el punto de contacto de la ventana.
64 a 65	Punto cero para las instrucciones de gráficos (CLIP OFFSET).

Todos los llamados AES, Line-A y VDI directos al igual que PUT, GET y BITBLT no respetan el punto cero.

La estructura de la inscripción del atributo de la ventana se conforma bit a bit. Cada bit representa una parte de la ventana. Cuando se fija un bit, la ventana posee el correspondiente elemento de ventana:

Bit	Elemento de ventana correspondiente
0	Inscripción del título de la ventana.
1	Campo de cierre en la parte superior izquierda.
2	Campo completo en la parte superior derecha.
3	Línea de cabecera, con la cual se puede desplazar la ventana.
4	Línea info debajo de la línea de título.
5	Campo de modificación del tamaño en la parte derecha inferior
6	Flecha señalando hacia arriba.
7	Flecha señalando hacia abajo.
8	Regulador vertical a la derecha.
9	Flecha señalando hacia la izquierda.
10	Flecha señalando hacia la derecha.
11	Regulador horizontal abajo.

Ejemplo:

```
' Es posible simular OPENW #n,x,y,w,h,attr por medio de WINDTAB
' En la versión 2.0 solo era posible de esa manera
```

```
'
OPENW #1,100,120,200,70,&HFFF
'
' entspricht
'
DPOKE WINDTAB+2,&HFFF
DPOKE WINDTAB+4,100
DPOKE WINDTAB+6,120
DPOKE WINDTAB+8,200
DPOKE WINDTAB+10,70
OPENW 1
'
' oder
'
WINDTAB(1,1)=&HFFF
WINDTAB(1,2)=100
WINDTAB(1,3)=120
WINDTAB(1,4)=200
WINDTAB(1,5)=70
OPENW 1
```

Otros casos

```
RC_INTERSECT(x1,y1,w1,h1,x2,y2,w2,h2)

 x1,y1,w1,h1:  iexp
 x2,y2,w2,h2:  ivar
```

La función RC_INTERSECT (rectangle intersection) sirve para determinar si se superponen dos rectángulos (dados por las coordenadas del canto izquierdo superior (x,y) y por el ancho w y la altura h).

Cuando los rectángulos se cortan, se retorna TRUE y x2,y2,w2,h2 contienen después de haberse llamado la función las coordenadas y la superficie de corte.

Si no resulta ninguna superficie de corte, se retorna FALSE y x2,y2,w2,h2 contienen las coordenadas de un rectángulo, que se encuentra entre los dos rectángulos indicados. El ancho w2 o la altura h2 son entonces negativos o igual a cero.

Esta función se aplica generalmente para el tratamiento del 'Redraw' en eventos GEM.

Ejemplo:

```
BOX 100,100,400,300
x=200
y=200
w=300
h=150
BOX x,y,x+w,y+h
'
IF RC_INTERSECT(100,100,300,200,x,y,w,h)
  PBOX x,y,x+w,y+h
ENDIF
```

--> Se dibujan dos rectángulos y la superficie de corte se representa

en negro.

RC_COPYs_adr,sx,sy,w,h,TOd_adr,dx,dy[,m]

s_adr,d_adr,sx,sy,w,h,dx,dy,m: iexp

La instrucción RC_COPY hace posible copiar rectángulos de la pantalla entre varias páginas de la pantalla. Los parámetros s_adr y d_adr contienen la dirección de la pantalla de origen (source) y la pantalla de destino (destination). Las coordenadas del canto izquierdo superior así como el ancho y la altura del rectángulo a copiar se transfieren en sx,sy,w y h. Las coordenadas del canto superior izquierdo del rectángulo de destino son dx y dy. Opcionalmente se puede indicar un modo de conexión (0 a 15, comparar PUT). Se preselecciona para el uso el valor 3 para m.

Ejemplo:

```
FILESELECT "\*.*","",f$
IF EXIST(f$)
  OPEN "i",#1,f$
  bild$=INPUT$(32000,#1)
  s_adr%=V:bild$
  d_adr%=XBIOS(2)
  '
  FOR i%=1 TO 1000
    RC_COPY s_adr%,RAND(10)*64,RAND(10)*40,64,40 TO d_adr%,RAND(10)*64,RAND(1
0)*40
  NEXT i%
  '
  CLOSE #1
ENDIF
```

--> Se carga un archivo en el formato de la pantalla (32000 byte) y se representa en fragmentos de pantalla al azar.

ALERTsym,text$,default,button$,elección

sym,default: iexp
text$,button$: sexp
elección: avar

La instrucción ALERT deja visualizar un Alert-box en la pantalla. En la expresión sym se fija, qué símbolo debe figurar en la mitad izquierda del Alert-Box. Vale:

0 --> ningún símbolo.
1 --> signo de exclamación.
2 --> signo de interrogación.
3 --> cartel de stop.

En la expresión string text$ figura el texto, que debe aparecer en el Alert-box. Se permiten como máximo 4 líneas de máximo 30 caracteres cada una. Las líneas se separan entre sí con un I. Las líneas que superan los 30 caracteres son cortadas.

La expresión default del botón de selección del Alert-box es el llamado Default-button. Este botón se puede elegir no solo con la tecla del ratón, sino también pulsando la tecla Return o Enter. Si no

existe un button con el número default, entonces no se puede abandonar el Alert-box con Return o Enter, sino solo por medio del ratón.

La expresión string button$ contiene el texto del button del menú. El largo máximo del texto por button es de 8 caracteres. Los textos de los diferentes buttons se separan con I.

En la variable elección se retorna el número del button elegido (comparar FORM_ALERT).

Ejemplo:

```
ALERT 1,"Elija|un botón !",1,"izquierdo|derecho",a%
ALERT 0,"Ha elegido el botón "+STR$(a%)+".",0," Ok ",a%
```

--> Aparece un Alert-Box con dos buttons. Después de elegir un button aparece un segundo box, que indica qué button ha sido seleccionado en el primer box. Este segundo Alert-box no posee ningún símbolo ni ningún Default-button.

FILESELECTcamino$,default$,nombre$

camino$,default$: sexp
nombre$: svar

Esta instrucción hace aparecer un File-Select-Box en la pantalla, con el cual se puede seleccionar un archivo del índice de un disco (RAM-Disk, disco rígido, etc).

En la expresión camino$ figura el nombre de la disquetera y de la carpeta, cuyo índice se ha de presentar. Si no se indica disquetera, se elige la disquetera actual. default$ es el nombre del archivo, que ha de aparecer en la parte izquierda del File-Select-Box como preseleccionado. En nombre$ se encuentra el nombre del archivo elegido, después de activarse el Ok-button del File-Select-Button. Si se eligió el button de interrupción, nombre$ contiene un string nulo.

El formato de camino$, default$ y nombre$ equivale a las convenciones del sistema jerárquico de archivos, que se ha descripto en el capítulo "Ingreso general de datos y salida general de datos", apartado "Administración de archivos".

(Comparar FSEL_INPUT.)

Ejemplo:

```
FILESELECT "A\*.PRG","GFABASIC.PRG",nombre$
IF nombre$=""
  PRINT "Usted ha elegido el botón de interrupción."
ELSE IF RIGHT$(nombre$)="\"
  PRINT "Usted ha elegido el botón OK sin indicar el archivo."
ELSE
  PRINT "Usted ha elegido el archivo: ";nombre$;"."
ENDIF
```

--> Se visualiza un File-Select-Box y se analiza la elección que tomó el usuario.

10 - Rutinas de sistema

GEMDOS, BIOS y XBIOS

```
GEMDOS(nr[,x,y...])
BIOS(nr[,x,y...])
XBIOS(nr,[,x,y...])
```

nr, x, y: iexp

Estas tres funciones permiten llamar rutinas GEMDOS, BIOS y XBIOS. Con esta finalidad se transfieren en cada caso el número de función nr y la lista de parámetros. Para poder transferir los parámetros en el formato de variable correcto, se puede fijar un W: o una L: delante de los parámetros. Los parámetros caracterizados con W: y aquellos sin caracterización se transfieren entonces como enteros de simple densidad (16 bit), los parámetros caracterizados con L: como enteros de doble densidad (32 bit).

Para esto ver el anexo.

Ejemplos:

```
IF GEMDOS(17)
  PRINT "Impresora lista"
ELSE
  PRINT "Impresora no está lista"
ENDIF
```

--> Examina si está lista una impresora en la interfase paralela (GEMDOS(17) retorna el estado de salida de la interfase paralela).

```
REPEAT
UNTIL BIOS(11,-1) AND 4
```

--> Espera hasta que se pulsa la tecla Control (BIOS(11,-1) retorna el estado de las teclas de conmutación del teclado).

```
CIRCLE 320,200,180
BMOVE XBIOS(2),XBIOS(2)+16000,16000
```

--> Dibuja un círculo, copia su mitad superior en la mitad inferior de la pantalla (XBIOS(2) retorna la dirección, en la cual comienza la memoria física de la pantalla).

```
L:x
W:x
```

x: iexp

Estas dos funciones sirven para transferir expresiones numéricas como enteros de simple densidad (2 byte,W:) o de doble densidad (4 byte,L:) cuando se llaman funciones del sistema operativo y rutinas C. Si no figura ninguna caracterización se usa el formato palabra.

Ejemplo:

```
CLS
FOR i% = 1 TO 100
```

```
  BOX RAND(639),RAND(399),RAND(639),RAND(399)
NEXT i%
PBOX 0,0,639,399
DIM screen_2|(32255)
phys_base%=XBIOS(2)
old_screen%=phys_base%
log_base%=V:screen_2|(0)+255 AND &HFFFFFF00
REPEAT
  IF MOUSEK=1
    ~XBIOS(5,L:log_base%,L:phys_base%,-1)
    SWAP log_base%,phys_base%
  ENDIF
  PLOT MOUSEX,MOUSEY
UNTIL MOUSEK=2
~XBIOS(5,L:old_screen%,L:old_screen%,-1)
```

--> Cuando se pulsa la tecla izquierda del ratón conmuta entre dos páginas de la pantalla. La pulsación de la tecla derecha del ratón conduce a la detención del programa (ver también XBIOS). Además se grafica en ambas páginas de la pantalla en la posición del ratón.

Llamados de Line-A

ACLIP, PSET, PTST, ALINE, HLINE, ARECT, APOLY, ACHAR, ATEXT

En el siguiente apartado se discuten las instrucciones que básicamente equivalen a instrucciones que ya han sido tratadas en el capítulo sobre representaciones gráficas. Aunque las impresiones gráficas son en parte notoriamente más rápidas a través de los llamados de Line-A y usan una sintaxis algo diferente. Delante de las salidas gráficas de Line-A debería fijarse siempre un clipping mediante ACLIP, ya que sino se pueden sobrescribir zonas de la memoria que se encuentran fuera de la memoria de la pantalla. Sino también se usa el clipping de cualquier instrucción anterior para gráficos (VDI o AES).

Un llamado de VDI modifica el clipping conectado con ACLIP. Las instrucciones de Line-A son independientes de las instrucciones VDI-DEFxxx.

Con respecto a la elección del color: Los colores indicados en las rutinas de Line-A equivalen a los números del registro de colores de hardware (como SETCOLOR) y no a los números que emplea el VDI (COLOR).

ACILPflag,xmin,ymin,xmax,ymax

flag,xmin,ymin,xmax,ymax: iexp

Esta instrucción posibilita la determinación de un rectángulo de limitación, o sea las impresiones gráficas a través de Line-A se limitan a este fragmento rectangular de la pantalla. En xmin,ymin,xmax e ymax figuran las coordenadas del extremo izquierdo superior y derecho superior del rectángulo de clipping. En flag<>0 se conecta y desconecta el clipping. ACLIP no vale (lamentablemente) para PSET, PTST, ALINE, HLINE y BITBLT.

```
PSETx,y,f

x,y,f:  iexp
```

PSET equivale a la instrucción PLOT y posibilita la fijación de puntos en las coordenadas x,y en el color f. f puede asumir en este caso, según la resolución, un valor entre 0 y 15.

Ejemplo:

```
FOR i%=0 TO 199 STEP 2
  PSET i%,i%,1
NEXT i%
```

--> Fija puntos con un intervalo de dos pixel desde 0,0 hasta 199,199.

```
PTST(x,y)

x,y:  iexp
```

La función PTST equivale a la función POINT. Retorna el color,que se presenta en la posición x,y de la pantalla.

Ejemplo:

```
PSET 100,100
x=PTST(200,100)
PRINT x,PTST(100,100)
```

--> Indica los colores, que se presentan en las posiciones de la pantalla 200,100 y 100,100.

```
ALINEx1,y1,x2,y2,f,lm,m

x1,y1,x2,y2,f,lm,m:  iexp
```

ALINE equivale a la instrucción LINE, en este caso contienen x1,y1, x2,y2 las coordenadas del punto inicial y final de la línea. La expresión f contiene la información del color y puede tener valores entre 0 y 15 según la resolución (ver COLOR). En lm se transfiere la información de bits del diseño de líneas deseado. Se trata de una máscara de 16 bit de longitud, en la cual cada bit (en el modo monocromático) fijado corresponde a un punto a dibujar.

El parámetro m fija el modo gráfico y puede tener valores entre 0 y 3. Los significados en este caso son los siguientes:

```
0 --> Reemplazar
1 --> Transparente
2 --> Invertir
3 --> Invertido transparente
```

Ejemplo:

```
ym%=INT{L~A-4}-1
FOR i%=0 TO 199
  ALINE i%,0,i%,ym%,1,&HFFFF,0
NEXT i%
```

--> Dibuja líneas verticales a lo largo de toda la altura de la pantalla desde la coordenada x 0 hasta la 199.

HLINEx1,y,x2,f,m,dir,anz_diseño

x1,x2,y,f,m,adr,anz_diseño: iexp

HLINE equivale a la instrucción LINE, aunque con esta instrucción solo se pueden dibujar líneas horizontales. En este caso contienen x1 y x2 las dos coordenadas x e y la coordenada y de la línea a dibujar. La expresión f contiene la información del color y puede tener valores entre 0 y 15 según la resolución (ver COLOR). El parámetro m fija el modo de representación gráfica y equivale a ALINE.

Dir es la dirección de un campo, en el cual figuran las informaciones de bits para el diseño de la línea (cada 16 bit). Anz_diseño es una máscara,con la cual está conectada la coordenada Y para servir como índice en la tabla de diseños de línea. Por eso anz_diseño debería ser menor en uno que una potencia de dos (0,1,3,7,15,...).

Ejemplo:

```
ACLIP 1,0,0,639,399
'
muster%=&X1111111111111111101010101010101
z%=V:muster%
'
FOR i%=0 TO 199
  HLINE 0,i%,639,1,0,z%,1
NEXT i%
```

--> En la variable diseño% se depositan dos diseños de líneas de 16 bit. El último parámetro HLINE es por lo tanto 1 (2 diseños). Las líneas dibujadas con HLINE usan entonces alternadamente los dos diseños de líneas de 16 bit que figuran en diseño%.

ARECTx1,y1,x2,y2,f,m,dir,anz_diseño

x1,y1,x2,y2,f,m,dir,anz_diseño: iexp

ARECT equivale a PBOX. En este caso x1,y1 y x2,y2 son los cantos enfrentados del rectángulo. Los parámetros f,m,dir y Anz_diseño tienen el mismo significado que en HLINE.

Ejemplo:

```
ACLIP 1,0,0,639,399
DIM muster&(1)
'
muster&(0)=&X1010101010101010
muster&(1)=&X0101010101010101
'
muster_adr%=V:muster&(0)
ARECT 100,100,200,200,1,0,muster_adr%,1
```

--> Dibuja un rectángulo relleno con un trama que equivale al fondo del Desktop

```
APOLYdir_pkt,anz_pky,y0TOy1,f,m,dir,anz_diseño

dir_pkt,anz_pkt,y0,y1,f,m,dir,anz_diseño:  iexp
```

APOLY tiene un cierto parecido con la instrucción POLYFILL, dibuja un trazo de línea invisible cerrado con n cantos y lo rellena con cualquier diseño. Para ello se transfiere en dir_pkt la dirección del arreglo, que contiene en forma alternada las coordenadas x e y de los puntos. El parámetro 'anz_pkt' contiene la cantidad de puntos. En y0 e y1 se indican las coordenadas y, las que determinan qué parte del polígono se ha de rellenar. Los parámetros f,m,dir,anz_diseño equivalen aquellos en HLINE.

Ejemplo:

```
DIM x&(9),y&(9),muster&(1)
FOR i%=0 TO 8
  x&(i%)=RAND(100)
NEXT i%
x&(9)=x&(0)
'
adr_kor%=V:x&(0)
muster&(0)=-1
adr_muster%=V:muster&(0)
'
ACLIP 1,0,0,200,200
'
FOR i%=1 TO 8
  APOLY adr_xkor%,9,0 TO 100,1,1,adr_muster%,0
NEXT i%
```

--> Dibuja el trazo cerrado de un polígono, que se rellena a continuación.

```
BITBLTdir%
BITBLTx%()

dir%:  iexp
x%():  arreglo de enteros de cuatro byte
```

La instrucción BITBLT llama a la rutina Line-A del mismo nombre. Contrariamente a esto, la instrucción BITBLT con tres arreglos de parámetros llama a una rutina de VDI. En la variante de la instrucción, en la cual se indica una dirección como parámetro, debe figurar a partir de esta dirección una tabla de 76 byte de largo. En la siguiente tabla figura el significado de los valores en esta zona de la memoria. En la columna Offset figura la distancia de los elementos de la tabla del comienzo de la tabla en bytes.

En la variante de la instrucción, en la que se debe indicar un arreglo de cuatro enteros de cuatro bytes de por lo menos 23 elementos figura un parámetro en cada elemento del arreglo. Usted puede obtener de la columna Indice de la siguiente tabla qué parámetro debe figurar en qué elemento del arreglo.

La rutina BITBLT modifica los parámetros marcados con un asterisco. Por eso resulta conveniente trabajar con la variante de la instrucción, que usa un arreglo como parámetro, ya que se produce una copia de los parámetros indicados allí, a la cual accede luego la

rutina BITBLT. Los elementos del arreglo no se modifican en este caso. En cambio, los elementos de las tablas se modifican al transferirse a una dirección.

Nombre	Indice	Offset	Significado
B_WD	00	00	Ancho de la matriz en pixel.
B_HT	01	02	Altura de la matriz en pixel.
PLANE_CT *	02	04	Cantidad de planos color.
FG_COL *	03	06	Color del primer plano.
BG_COL *	04	08	Color de fondo.
OP_TAB	05	10	Conexión lógica para cada combinación de pixel del primer plano y del fondo.
S_XMIN	06	14	x-offset en la matriz de origen.
S_YMIN	07	16	y-offset en la matriz de origen.
S_FORM	08	18	Dirección de la matriz de origen.
S_NXWD	09	22	Offset para la próxima palabra del mismo plano.
S_NXLN	10	24	Offset para la próxima línea de la matriz de origen.
S_NXPL	11	26	Offset para el próximo plano de color (2).
D_XMIN	12	28	x-offset en la matriz de destino.
D_YMIN	13	30	y-offset en la matriz de destino.
D_FORM	14	32	Dirección de la matriz de destino.
D_XNWD	15	36	Offset para la próxima palabra del mismo plano.
D_NXLN	16	38	Offset para la próxima línea de la matriz de destino.
D_NXPL	17	40	Offset para el próximo plano de color (siempre 2).
P_ADDR	18	42	Indicador en la tabla con diseños de relleno, cuando no es cero se conecta con AND.
P_NXLN	19	46	Offset para la próxima línea de la matriz de máscaras.
P_NXPL	20	48	Offset para el próximo color del diseño de relleno.
P_MASK	21	50	Máscara igual que en HLINE,
SPACE *	22	52	los siguientes 24 bytes sirven como zona de trabajo para el blitter.

Ejemplos:

```
DIM x%(1000)
'
FOR i%=0 TO 639 STEP 8
  LINE i%,0,639,399
  NEXT i%
'
GET 0,0,639,399,a$
mirrorput(0,0,a$)
'
PROCEDURE mirrorput(x%,y%,VAR x$)
  IF LEN(x$)>6      !solo si hay algo
    xx%=V:x%(0)
    a%=V:x$
    b%=INT{a%}
    h%=INT{a%+2}
    '
```

```
      INT{xx%}=1
      INT{xx%+2}=h%
      INT{xx%+4}=1
      INT{xx%+6}=1
      INT{xx%+8}=0
      {xx%+10}=&H3030303
      INT{xx%+14}=9999
      INT{xx%+16}=0
      {xx%+18}=a%+6
      INT{xx%+22}=2
      INT{xx%+24}=SHR(b%+16,4)*2
      INT{xx%+26}=2
      INT{xx%+28}=9999
      INT{xx%+30}=0
      {xx%+32}=XBIOS(3)
      INT{xx%+36}=2
      INT{xx%+38}=80
      INT{xx%+40}=2
      {xx%+42}=0     !pattadr
      INT{xx%+46}=0     !p_nxtln
      INT{xx%+48}=0     !p_nxpl
      INT{xx%+50}=0     !p_mask
      '
      ABSOLUTE i&,xx%+14
      ABSOLUTE di&,xx%+28
      '
      FOR i&=0 TO b%
        INT{xx%+4}=1
        di&=SUB(639,i&)
        BITBLT xx%
      NEXT i&
      '
    ENDIF
  RETURN

  --> Un ejemplo para BITBLTdir%.

  DIM x%(1000)
  '
  FOR i%=0 TO 639 STEP 8
    LINE i%,0,639,399
  NEXT i%
  '
  GET 0,0,639,399,a$
  mirrorput(0,0,a$)
  '
  PROCEDURE mirrorput(x%,y%,VAR x$)
    IF LEN(x$)>6       !solo si hay algo
      a%=V:x$
      b%=INT{a%}
      h%=INT{a%+2}
      '
      x%(0)=1
      x%(1)=h%
      x%(2)=1
      x%(3)=1
      x%(4)=0
      x%(5)=&H3030303
      x%(6)=9999
```

```
    x%(7)=0
    x%(8)=V:x$+6
    x%(9)=2
    x%(10)=SHR(b%+16,4)*2
    x%(11)=2
    x%(12)=9999
    x%(13)=0
    x%(14)=XBIOS(3)
    x%(15)=2
    x%(16)=80
    x%(17)=2
    x%(18)=0     !pattadr
    x%(19)=0     !p_nxtln
    x%(20)=0     !p_nxpl
    x%(21)=0     !p_mask
    '
    FOR i%=0 TO b%
      x%(6)=i%
      x%(12)=639-i%
      BITBLT x%()
    NEXT i%
    '
  ENDIF
RETURN
```

--> Este es un ejemplo para BITBLTx%().

Ambas rutinas reflejan en la pantalla un fragmento leído con GET. Igual que el listing de BITBLT en el capítulo sobre Instrucciones Generales para Representaciones gráficas. En la rutina que usa BITBLTdir% se debe volver a fijar cada vez la cantidad de bitplanes. Cuál de las rutinas BITBLT se usará, es cuestión de gustos. En cuanto a la velocidad hay pocas diferencias.

ACHARcode,x,y,font,estilo,angulo

code,x,y,font,estilo,angulo: iexp

Con ACHAR se puede imprimir un único carácter con cualquier código del valor ASCII en el punto (x,y). La expresión numérica font puede asumir en este caso valores entre 0 y 2. A este valor se le asignó lo siguiente:

0 = 6x6 (letra Icon)
1 = 8x8 (letra normal en el modo color)
2 = 8x16 (letra normal monocromática)

Valores mayores para font se interpretan como direcciones font-header. Un font de este tipo debe presentarse en el formato, en el que será transformado por GDOS al cargarse, o sea en el formato Motorola con high-byte antes de low-byte.

Adicionalmente se pueden fijar el estilo del texto (0 a 31) y ángulo de salida (0,900,1800,2700). Estos valores equivalen a aquellos de la instrucción TEXT. Aunque contrariamente a la instrucción TEXT, las indicaciones de las coordenadas se refieren al canto superior de los caracteres.

```
ATEXTx,y,font,s$

x,y,font:  iexp
s$:  sexp
```

La instrucción ATEXT sirve para la salida de cadenas de caracteres en posiciones cualesquiera de la pantalla. A diferencia de ACHAR no se puede indicar en este caso ningún estilo de letra ni ángulo de salida especial. Los parámetros x,y y font son los mismos que en ACHAR.

Ejemplo:

```
a_clock                          ! llama Prozedure a_clock
'                                  por primera vez
EVERY 400 GOSUB a_clock          ! cada 2 segundos llamar a_clock
OPENW 0                          ! proteger linea superior
'                                ! para que no se escriba en ella
FOR i%=1 TO 100000
  PRINT USING " ###### ",i%;     ! devolver números
NEXT i%
'
PROCEDURE a_clock
  ACLIP 1,20,0,120,15            ! encenderClipping , sino vale el
  '                                del OPENW 0, d.h.: no se ve nada.
  ATEXT 20,0,2,TIME$             ! Indicar la hora
RETURN
```

--> Evidencia cada dos segundos la hora y escribe columnas de números.

L~A

La función L~A calcula la dirección básica de las variables Line-A (ver anexo).

Ejemplo:

```
PRINT INT{L~A
```

--> Da el número de los bitplanes en la resolución actual. Ver también los ejemplos bajo BITBLT.

Llamados de VDI

Las funciones de VDI se subdividen en siete zonas:

- funciones de control.
- funciones de salida.
- funciones de atributo.
- funciones de matrices.
- funciones de ingreso.
- funciones de información.
- escapes.

Se diferencian tres tipos de parámetros: ingreso, salida y aquellos, que se pueden usar tanto para el ingreso como para la salida. Estos parámetros se transfieren en cinco arreglos:

- **CONTRL** Control.
- **INTIN** Ingreso de enteros.

- **PTSIN** **Ingreso de coordenadas de puntos.**
- **INTOUT** **Salida de enteros.**
- **PTSOUT** **Salida de coordenadas de puntos.**

Los parámetros de ingreso figuran en:

CONTRL(0) Opcode.

CONTRL(1) Cantidad de puntos en el arreglo PTSIN.

CONTRL(3) Longitud del arreglo INTIN.

CONTRL(5) Identificación del opcode.

INTIN() Arreglo de ingreso de los valores enteros.

PTSIN() Arreglo de ingreso de las coordenadas de puntos.

Los parámetros de salida son:

CONTRL(2) Cantidad de puntos en el arreglo PTSOUT.

CONTRL(4) Longitud del arreglo INTOUT.

INTOUT() Arreglo de salida de los valores enteros.

PTSOUT() Arreglo de salida de las coordenadas de puntos.

Los parámetros de ingreso y salida son:

CONTRL(6) Caracterización de los aparatos.

CONTRL(7-n) Informaciones dependientes del opcode.

Los valores, que se deben inscribir en los respectivos campos cuando se llama una función VDI, difieren según la función.

CONTRL
INTIN
PTSIN
INTOUT
PTSOUT

Estas funciones dan las direcciones de los bloques de parámetros de VDI. Con un índice entre paréntesis se accede a los elementos de estos campos:

CONTRL --> Dirección del campo de control del VDI.
INTIN --> Dirección del campo de Input de enteros del VDI.
PTSIN --> Dirección del campo de Input de las coordenadas de puntos del VDI.
INTOUT --> Dirección del campo de output de enteros del VDI.
PTSOUT --> Dirección del campo de output de coordenadas de puntos del VDI.

Estos campos contienen los parámetros para los llamados del VDI.

Por ej., CONTRL(2) equivale al DPOKE CONTRL +4 o bien al

DPEEK(CONTRL+4). Los otros campos también están ordenados por palabras.

VDISYS[opc[,c_int,c_pts[,subopc]]]

opc,c_int,c_pts,subopc: iexp

Con la instrucción VDISYS se llama la función de VDI con el número opc. Si no se indica opc, se debe inscribir el número de función al igual que los demás parámetros con DPOKE CONTRL , opc o CONTRL(0)=opc en el campo de control.

En los parámetros c_int y c_pts se puede indicar la cantidad de valores en el campo de Input de enteros y en el campo de Input de coordenadas de puntos. Luego no se deben inscribir más estos valores en el campo de control. El parámetro opcional subopc contiene el subopcode de la rutina de VDI a llamar. Este solo se debe indicar en algunas rutinas de VDI, por ej. en las rutinas de escape. Los parámetros c_int, c_pts y subopc se aplican en determinadas inscripciones de campos de CONTRL.

Vale lo siguiente:

```
opc      -->  CONTRL(0)
c_int    -->  CONTRL(3)
c_pts    -->  CONTRL(1)
subopc   -->  CONTRL(5)
```

Ejemplos:

```
CONTRL(1)=3
CONTRL(5)=4
PTSIN(0)=320
PTSIN(1)=200
PTSIN(4)=190
VDISYS 11

PTSIN(0)=320
PTSIN(1)=200
PTSIN(4)=190
VDISYS 11,0,3,4

PCIRCLE 320,200,190
```

--> Dibuja tres veces un círculo relleno en la pantalla.

```
VDISYS 5,0,0,13
PRINT " Invertido "
VDISYS 5,0,0,14
PRINT " Normal "
```

--> Escribe el texto 'invertido' invertido y el texto 'Normal' no invertido.

VDIBASE

La variable de sistema VDIBASE contiene la dirección, a partir de la cual se debe deponer el parámetro actual de la versión GEM para la administración de las rutinas de VDI. La estructura de la zona de

memoria puede diferir en versiones futuras . Por eso se suprime un ejemplo en este caso. El GFA-BASIC contiene VDIBASE , para darle la oportunidad a los programadores de poder acceder a todos los parámetros de VDI.

WORK_OUT(x)

x: iexp

Esta función da los valores, que figuran en el llamado de las funciones de VDI OPEN WORKSTATION en intout(0) hasta intout(44) y en ptsout(0) hasta ptsout(1). Solo que el índice corre desde 0 hasta 56 (ver anexo) cuando se emplean las funciones WORK_OUT.

Ejemplo:

```
raster_breite&=WORK_OUT(0)
raster_hoehe&=WORK_OUT(1)
PRINT raster_breite&,raster_hoehe&,WORK_OUT(10)
```

--> Retorna los números 639 y 399 para el ancho y la altura de la matriz (en el modo monocromático) y un 1 para la cantidad de símbolos disponibles (WORK_OUT(10)).

Rutinas de VDI especiales y GDOS

Se puede disponer de las siguientes funciones de workstation VDI y para solicitar algo si en el booting se incluyó el programa GDOS (a partir de Release 1.0) y si existe un archivo ASSIGN.SYS válido.

El GDOS (Graphics Device Operating System) independiente de los aparatos contiene funciones gráficas y trabaja con drivers dependientes de aparatos para distintos equipos de salida (pantalla, impresora, graficador (plotter), metaarchivo,etc.).

El archivo ASCII ASSIGN.SYS contiene todos los datos necesarios sobre la configuración de aparatos usada. En este archivo se deben registrar todos los drivers de aparatos y los símbolos, eventualmente se debe indicar además el camino de acceso para driver de aparatos, cuando éstos no se encuentran en la disquetera A:. En este caso se debe respetar la siguiente sintaxis:

```
id          DRIVER.SYS;
;           Observaciones
            orac1.FNT
            orac2.FNT
            ...
            oracn,FNT
```

id Contiene un número entre 1 y 32767. Así se fija el tipo de driver del aparato, valiendo la siguiente asignación:

```
01...Pantalla
11...Graficador (Plotter)
21...Impresora
31...Metafile
41...Cámara
51...Bandeja de gráficos
```

Cuando los drivers de los aparatos se encuentran en la disquetera C: del ordenador GEMSYS, entonces el archivo ASSIGN.SYS correspondiente es el siguiente:

```
path = c:\gemsys\
01p screen.sys;    driver de la pantalla proveniente del ROM, por eso
                   figura una p detrás de id
ATSS10.FNT
ATSS!@.FNT
02p screen.sys;    driver para baja resolución
ATSS10.FNT
ATSS12.FNT
03p screen.sys;    driver para resolución intermedia
ATSS10CG.FNT
ATSS12CG.FNT
04p screen.sys;    driver para resolución alta
ATSS10.FNT
ATSS12.FNT
21 fx80.sys;       driver de la impresora para el FX-80 y compatibles
ATSS10EP.FNT
ATSS12EP.FNT
31 META.SYS        metafile-driver
ATSS10MF.FNT
ATSS12MF.FNT
```

En este caso los drivers de la pantalla SCREEN.SYS solo son inscripciones dummy, exigidas así por el control de sintaxis del GDOS. Los id's (2,3 y 4) solo significan que a cada resolución se le asignan los correspondientes fonts. Por eso el llamado para abrir una workstation virtual de la pantalla es el siguiente: V_OPNVWK(XBIOS(4)+2). Este llamado lo ejecuta el GFA-BASIC internamente.

GDOS?

GDOS? da TRUE cuando GDOS está cargado (a partir de Release 1.0) y da FALSE, cuando GDOS no es residente.

Ejemplo:

```
IF NOT GDOS?
  ALERT 1,"No he encontrado GDOS !",1,"Cancelar",r%
  END
ENDIF
```

V~H

En las siguientes funciones de control de VDI, implementadas en el GFA-BASIC, figura hd para VDI-handle y id para la caracterización de un aparato de salida. En relación con esto se deben respetar las siguientes funciones:

V~H Da el VDI-handle usado internamente por el GFA-BASIC (por ej. PRINT V~H).

V~H=x Fija el VDI-handle usado internamente (y al cual se puede acceder a través de CONTRL(6)) en el valor x.

V~H=-1 Fija el VDI-handle usado internamente (y al cual se puede acceder a través de CONTRL(6)) en el valor, que dio el V_OPNVWK usado internamente.

V_OPNWK(id)
V_OPNWK(id,1,1,1,1,1,1,1,1,1,2)
V_CLSWK()

id: iexp

Los números 1 o bien 2 muestran los valores default para el ajuste de los parámetros del VDI, también se pueden modificar (comparar WORK_OUT).

La función V_OPNWK (open workstation) da el handle hd para la caracterización del aparato indicada id. Además se pueden solicitar otras informaciones sobre el aparato conectado con INTOUT() y PTSOUT() (comparar VDISYS en el apartado de rutinas de sistema).

La función V_CLSWK (close workstation) abre la estación de trabajo abierta (con V_OPNWK) y escribe todos los contenidos de los buffer de esta estación. Además, se ejecuta V~H=-1.

Ejemplo: (comparar en V_CLRWK).

V_OPNVWK(id)
V_OPNVWK(id,1,1,1,1,1,1,1,1,1,2)
V_CLSWK(id)

id: iexp

La función V_OPNVWK (open virtual workstation) abre un driver de pantalla virtual y da el handle para la caracterización indicada del aparato id (comparar V_OPNWK).

La función V_CLSVWK (close virtual workstation) cierra una workstation virtual abierta (con V_OPNVWK). Además, se ejecuta V~H=1.

V_CLRWK()
V_UPDWK()

La función V_CLRWK (clear workstation) borra el buffer de salida. En este caso se borran por ej. el buffer de la pantalla o de la impresora.

En la salida de gráficos en la impresora se recolectan todos los comandos en un buffer. La función V_UPDWK (update workstation) evidencia estas instrucciones de gráficos de buffer en el aparato conectado. En cambio en la pantalla se ejecutan inmediatamente todas las instrucciones de gráficos.

Ejemplo:

```
RESERVE 25600                          ! reservar suficiente memoria
'
handle%=V_OPNWK(21)                    ! Determinar número de periférico
'
IF handle%=0
  ALERT 3,"Error de instalación !",1,"Cancelar",r%
  RESERVE
  END
ENDIF
```

```
'
x_res%=INTOUT(0)                    ! determinar resolución en x,
y_res%=INTOUT(1)                    ! del equipo conectado
'
V~H=handle%                         ! fija el Handle de VDI
'                                     al número del periférico
~V_CLRWK()                          ! Borrar Buffer
'
CLIP 0,0,x_res%,y_res%
BOX 0,0,x_res%,y_res%
LINE 0,0,x_res%,y_res%
LINE 0,y_res%,x_res%,0
'
~V_UPDWK()                          ! ejecutar los comandos gráficos
~V_CLSWK()
'
RESERVE                             ! liberar la memoria
```

--> Cuando GDOS está activo se visualiza en una impresora acoplada, un rectángulo, en el cual se dibujaron las diagonales.

VST_LOAD_FONTS(x)
VST_UNLOAD_FONTS(x)

x: iexp

La función VST_LOAD_FONTS carga los símbolos adicionales indicados en ASSIGN.SYS, si alcanza el espacio de memoria disponible y retorna la cantidad de letras realmente cargadas. Si no se dispone de más fonts, se retorna cero.

El parámetro x debería ser igual a cero, ya que son posibles ampliaciones en otras versiones GEM. Ante todo es importante, reservar con RESERVE suficiente espacio de memoria para los símbolos adicionales.

La función VST_UNLOAD_FONTS elimina de la memoria los símbolos cargados anteriormente con VST_LOAD_FONTS . Para el parámetro vale lo mismo que en VST_LOAD_FONTS.

Ejemplo:

```
RESERVE 25600
'
anz_fonts%=VST_LOAD_FONTS(0)      ! Número de set de letras ?
face%=VQT_NAME(anz_fonts%,font$) ! Index y nombre de los sets
'
FOR i%=1 TO anz_fonts%
  DEFTEXT ,,,face%
  TEXT 80,80,"Este es el "+font$+" Font."
  ~INP(2)
NEXT i%
'
~VST_UNLOAD_FONTS(0)              ! borrar fonts
'
RESERVE
```

--> En la pantalla se visualiza el nombre de los símbolos cargados en este fonts.

VQT_EXTENT(text$[,x1,y1,x2,y2,x3,y3,x4,y4])

text$: sexp
x1,y1,x2,y2,x3,y3,x4,y4: ivar

La función VQT_EXTENT da las medidas de un rectángulo, que incluye el string text$ transferido. Opcionalmente puede tener lugar el retorno en las variables x1,y1 a x4,y4 o solo en PTSOUT(0) a PTSOUT(7). Los cantos se enumeran en sentido contrario al reloj.

Canto	Posición
x1,y1	Canto izquierdo inferior.
x2,y2	Canto derecho superior.
x3,y3	Canto derecho superior.
x4,y4	Canto izquierdo superior.

VQT_NOMBRE(i,font_nombre$)

i: ivar
font_nombre$: svar

La función VQT_NOMBRE da la caracterización de la secuencia de símbolos cargada con el número i y registra en la variable string font_nombre$ el nombre de este font.

Ejemplo:

```
RESERVE 2560
'
anz_fonts%=VST_LOAD_FONTS(0)
face%=VQT_NAME(anz_fonts%,font$) ! Index und nombre de los sets
h%=12                            ! Altura de texto
s$="Beispieltext"
x0%=80                           ! Coordinadas
y0%=80                           ! para s$
DEFTEXT 1,0,0,h%,face%
'
~VQT_EXTENT(s$,x1%,y1%,x2%,y2%,x3%,y3%,x4%,y4%)
'
GRAPHMODE 4
TEXT x0%,y0%,s$
PBOX x0%+x1%,x0%+y1%-h%-1,x0%+x3%,y0%+y3%-h%
'
~VST_UNLOAD_FONTS(0)             ! Borrar Fonts
'
RESERVE
```

--> El string s$ sale en la pantalla en forma invertida en el punto x0,y0.

El llamado de rutinas escritas en otros lenguajes

Los comandos descriptos en este apartado sirven para llamar subrutinas, que están escritas en C o en assembler.

C:dir([x,y,...])

dir: avar (por lo menos 32 bit, preferentemente dir%)
x,y: iexp

La función C llama un subprograma escrito en C o en assembler que figura en la dirección dir. Los parámetros x,y...se pueden transferir entre paréntesis . La transferencia de parámetros ocurre igual que en C. Los parámetros se pueden transferir como palabras de doble densidad de 31 bits o de simple densidad de 16 bits, estando preseleccionada la transferencia como valor de 16 bits. Se pueden transferir palabras de doble densidad si delante de los correspondientes parámetros se fija una L:. Cuando se llama esta función se colocan sobre el stack primero la dirección de retorno y luego los parámetros . Así por ej.:

```
VOID C:dir%(L:x,W:y,z)
```

conduce a la siguiente situación en el stack:

```
  (sp) --> dirección de retorno
 4(sp) --> x (4 bytes)
 8(sp) --> y (2 bytes)
10(sp) --> z (2 bytes)
```

El valor retornado por la función es el contenido del registro d0 cuando retorna de la rutina (que debe realizarse con RTS).

Ejemplo:

El programa assembler usado aquí rellena un espacio de la memoria (por ej. un arreglo de enteros) a partir de una dirección determinada con números (enteros de doble densidad) entre 0 y n.

```
206F0004        move.l      4(sp),a0      ;dirección inicial
202F0008        move.l      8(sp),d0      ;cantidad de valores
7200            moveq,l     #0,d1         ;contador
6004            bra.s       ct_2          ;salto al loop
20C1      ct_1:move.l       d1,(a0)+      ;loop, escribir valor
5281            addq.l      #1,d1         ;incrementar contador
B081      ct_2:cmp.l        d1,d0         ;listo?
64F8            bcc.s       ct_1          ;no,seguir -->
4E75            rts                       ;retorno al GFA-BASIC
```

El programa BASIC es el siguiente:

```
FOR i%=1 TO 11
  READ a%
  asm$=asm$+MKI(a%)
NEXT i%
DATA $206F,$0004,$202F,$0008,$7200,$6004,$20C1,$5281,$B081,$64F8,$4E75
'
DIM x%(10000)
asm%=V:asm$
~C:asm%(L:V:x%(0),L:10000)
PRINT "por ej. x%(12): ";x%(12)
```

--> Llena el arreglo x%() con los números 0 a 10000.La ejecución de la función C: equivale en esta aplicación a las siguientes instrucciones:

```
FOR i%=1 TO n%
  x%(i%)=i%
NEXT i%
```

Otra posibilidad de insertar un programa assembler en un programa del GFA-BASIC 3.0 está dada por la instrucción INLINE. Primero se debe producir un archivo, en el cual figura el programa assembler indicado anteriormente, por ej. con (de acuerdo con la estructura de repetición READ-DATA recién mencionada):

```
BSAVE "COUNT.INL",V:asm$,22
```

Luego escribir el siguiente programa:

```
INLINE asm%,22
DIM x%(10000)
~C:asm%(LLV:x%(0),L:10000)
```

Luego se debe pulsar la tecla Help en la línea en la que figura INLINE, seleccionar Load y cargar COUNT.INL. Entonces se puede almacenar este programa con Save.

MONITOR[x]

x: iexp

Esta instrucción sirve para llamar subrutinas assembler o programas Debug u otros programas auxiliares. Con esta finalidad se debe "torcer" el illegal instruction vector (dirección 16) a la dirección de la subrutina. La instrucción MONITOR genera entonces una illegal instruction exception, que conduce a la iniciación del subprograma, que debe terminar con RTE (return from exception). El parámetro x es un valor de transferencia, que se escribe en el registro d0.

Un ejemplo de aplicación se puede usar en un programa como el que se describió en C:, en un debugger. Para ello, se carga e inicia el GFA-BASIC por el debugger. Luego, después de ingresar el programa, se debe insertar directamente detrás de la instrucción INLINE: MONITOR asm%. Después de la iniciación del programa aparecerá un mensaje por parte del debugger debido a una "Illegal instruction". Ahora se puede desasemblar o editar la zona a la cual señala d0. A continuación se puede, eventualmente después de pulsar Shift-Alternate-Control, continuar la ejecución BASIC con la instrucción del debugger.

CALLdir([x,y,...])

dir: avar (por lo menos 32 bit, preferentemente dir%)
x,y: iexp

Con la instrucción CALL también se pueden llamar rutinas assembler o C. La dirección 'dir' es la dirección a partir de la cual se encuentra el programa assembler. Existe la posibilidad de transferir una lista de parámetros. Después del llamado de CALL, se encuentra la dirección de retorno en el stack (la rutina assembler debe terminar con RTS). Luego figura la cantidad de parámetros transferidos como valor de 16 bit y finalmente la dirección, a partir de la cual figuran los parámetros en la memoria como valores de 32 bit. Todos los parámetros se interpretan como palabras de doble densidad (32 bit)

Aquí también existe la posibilidad de transferir strings como parámetros. En este caso el valor de transferencia es la dirección inicial del string.

Estructura del stack

```
 (sp) --> dirección de retorno
4(sp) --> cantidad de parámetros (16 bit)
6(sp) --> dirección del campo de parámetros (32 bit)
```

RCALLdir,reg%()

reg%(): nombre arreglo de enteros de 4 bytes
dir: iexp

La instrucción RCALL hace posible asignar valores a los registros antes de comenzar la rutina assembler y solicitar los contenidos de los registros después de la ejecución de la rutina. Para ello sirve el arreglo reg%(), cuyos elementos deben ser del tipo entero de 4 bytes y que debe tener por lo menos 16 elementos. Antes de la iniciación de la rutina assembler se copian las inscripciones de este arreglo en los registros y después de la ejecución de la rutina se escriben los contenidos de los registros en los respectivos elementos del arreglo. En este caso vale la siguiente asignación (con OPTION BASE 0):

Registro de datos	d0 a d7	en reg%(0) a reg%(7)
Registro de direcciones	a0 a a6	en reg%(8) a reg%(14)
User-stack-pointer	(a7)	en reg%(15) (solo retorno)

Ejemplo:

El assembler-listing espera en a0 la dirección de la memoria (lógica o física) de la pantalla. Luego invierte la pantalla entre las coordenadas y, que se indican en d0 y d1. En la pantalla de color se duplican las coordenadas.

```
sub  d0,d1          ;cantidad de líneas
mulu #20,d1         ;criterio de interrupción del loop
subq  #1,d1         ; "
mulu  #80,d0        ;cantidad de bytes que debe saltear
dir.l d0,a0         ;dirección a partir de la cual se invierte
loop                ;comienzo del loop
not.l(a0)+          ;invierto 4 bytes
dbra d1,loop        ;disminuyo y ramifico
rts                 ;de vuelta al GFA-BASIC
```

El programa de GFA-BASIC dice:

```
DO
  READ a%
  EXIT IF a%=-1
  A$=A$+MKI$(a%)
LOOP
DATA 37400,49916,20,21313
DATA 49404,80,53696
DATA 18072,20937,65532,20085,-1
'
DIM r%(16)
zb2%=XBIOS(2)
```

```
HIDEM
FOR j%=1 TO 50
  FOR i%=0 TO 190  STEP 10
    r%(0)=i%
    r%(1)=399-i%
    r%(8)=xb2%
    RCALL V:a$,r%()
  NEXT i%
NEXT j%
```

--> El programa produce un juego gráfico. En este caso, al igual que en C, por supuesto también se puede emplear INLINE.

EXECmod,nom,cmdl,envs
EXEC(mod,nom,cmdl,envs)

mod: iexp
nom,cmdl,envs: sexp

La instrucción EXEC se puede aplicar tanto como comando como función. Sirven para cargar e iniciar programas de disco que después de finalizados vuelven al programa que los solicita. Antes del llamado de EXEC se debe brindar espacio de memoria al programa que se solicita (ver ejemplo). El parámetro mod indica el modo del llamado. En este caso significa:

0 --> cargar e iniciar el programa.
3 --> solo cargar el programa.

En la expresión nom figura el nombre del programa a cargar (y a iniciar). El formato de la indicación del nombre equivale a las reglas del sistema jerárquico de archivos, que se encuentran descriptos en el capítulo de Ingreso general de datos y salida general de datos.

La expresión cmdl contiene la línea de sentencia o comando, que se encuentra inscripta en la basepage del programa llamado. El primer carácter de la línea de sentencia contiene el largo de la misma (como máximo 127). En general se ignora este byte, salvo que sea cero. Por eso generalmente es suficiente insertar un símbolo dummy (*).

En envs figura el environment. Este es un string subdividido por CHR$(0). Para un programa C es por lo tanto una serie de strings, cuyo último string se caracteriza por un doble byte de ceros (lo que hace el GFA-BASIC por sí mismo). Este environment es usado por muchos programas shell, para almacenar variables y sus nombres (generalmente instrucción SET). Muchos compiladores también lo usan para fijar caminos de acceso o cosas similares.

Si se llama EXEC como función, entonces se obtiene el valor de retorno del programa o, para modo=3, la dirección de la basepage del programa solicitado. EXEC 3 solo tiene una aplicación útil en programas especialmente escritos con esta finalidad (overlays) o para debuggers.

Ejemplo:

```
FILESELECT "\*.PRG","",f$
RESERVE 100
SHOWM
```

```
a%=EXEC(0,f$,"","")
RESERVE
PRINT "Devuelta en el GFA-BASIC, valor de devolución ",a%
```

--> Permite llamar un programa (en la medida en que no requiera demasiado espacio de memoria) y vuelve al intérprete después de finalizado el programa solicitado. El valor de retorno, que da la función EXEC, también puede ser fijado por programas de GFA-BASIC a través de QUITn o SYSTEM.

11 - Bibliotecas AES

El siguiente capítulo ha de dar un panorama sobre las bibliotecas AES (AES = Application Environment Services). Una descripción detallada de estas rutinas excedería los límites de un manual, por eso se indicará en este lugar la amplia bibliografía sobre el tema GEM. En este capítulo se darán solo informaciones en forma comprimida. Las diferentes rutinas se presentan de acuerdo con el siguiente esquema:

En la primera línea figura el nombre de la correspondiente rutina de biblioteca, luego sigue una breve descripción de la función. A continuación se presenta el llamado completo de la función en la notación del GFA-BASIC 3.0 y se explica el significado de todas las variables usadas.

El capítulo concluye con algunos programas de ejemplo más extensos.

Antes de comenzar a describir las instrucciones de las bibliotecas AES, han de conocerse las principales estructuras de datos, con que trabajan las AES. Estas son las estructuras OBJEKT, TEDINFO, ICOBI, BITBLK, USERBLK y PARMBLK.

En primer lugar se dará la estructura de las instrucciones AES, que están contenidas en forma similar en el GFA-BASIC 2.0.

GCONTRL
ADDRIN
ADDROUT
GINTIN
GINTOUT
GB

Estas funciones calculan las direcciones de los bloques de parámetros AES. Estos son :

GCONTRL --> Dirección del arreglo de control AES.
ADDRIN --> Dirección del arreglo de Input de direcciones AES.
ADDROUT --> Dirección del arreglo de Output de direcciones AES.
GINTIN --> Dirección del arreglo de Input de enteros de AES.
GINTOUT --> Dirección del arreglo de Output de enteros de AES.
GB --> Dirección del arreglo de parámetros AES.

Estas funciones con el índice entre paréntesis acceden directamente a los campos correspondientes. GCONTRL(3) equivale a DPEEK(GCONTRL+6) o bien al correspondiente DPOKE.

En este caso tanto GCONTRL como GINTIN y GINTOUT están organizados por palabras, ADDRIN y ADDROUT en enteros de doble densidad.

O sea, ADDRIN(2)=x equivale a LPOKE ADDRIN + 2*4,x.

El campo GB no se puede usar con índices, ya que por razones de compatibilidad solo está a disposición del ST-BASIC; igual existe GINTIN, etc. El segundo entero de doble densidad ({GB + 4}) es la dirección del campo interno global del GEM.

GEMSYSnr

nr: iexp

Con la instrucción GEMSYS se puede llamar una rutina AES con su número de función. Los parámetros necesarios para esta rutina se deben dejar en los bloques de parámetros AES.

Ejemplo:

```
REPEAT
  GINTIN(0)=60
  GINTIN(1)=30
  GINTIN(2)=10
  GINTIN(3)=10
  GINTIN(4)=200
  GINTIN(5)=200
  GEMSYS 72
UNTIL MOUSEK
```

--> Dibuja un rectángulo que se mueve sobre la pantalla (72 es el número de la función GRAF_MOVEBOX).

Estructura-Objeto

Offset	Contenido	Tipo	Significado
00	ob_next	word	Señalador sobre el siguiente objeto.
02	ob_head	word	Señalador sobre el primer niño.(child)
04	ob_tail	word	Señalador sobre el último niño.
06	ob_type	word	Tipo del objeto.
08	ob_flags	word	Informaciones del objeto (ver más abajo)
10	ob_state	word	Estado del objeto (ver más abajo).
12	ob_spec	long	Señalador sobre otras informaciones (ver más abajo).
16	ob_x	word	Posición x del objeto.
18	ob_y	word	Posición y del objeto.
20	ob_w	word	Ancho del objeto.
22	ob_h	word	Altura del objeto.

El requerimiento de espacio de almacenamiento de un objeto comprende entonces estos bytes más otras estructuras de descripción eventualmente presentes, por ej. estructuras TEDINFO o BITBLK.

OB_NEXT Señalador sobre el siguiente objeto en el mismo plano o sobre el objeto madre, en el caso en que se trata del último elemento en el plano.

OB_HEAD Señalador sobre el primer niño del correspondiente objeto.

OB_TAIL Señalador sobre el último niño del correspondiente objeto.

Si no existen planos más profundos, los señaladores tienen el valor -1, también llamado NIL (not in list).

Los siguientes valores de OB_TYPE son posibles. De acuerdo con el valor de OB_TYPE, OB_SPEC es la dirección de diferentes estructuras de información.

OB_TYPE		OB_SPEC
G_TEXT	21	Dirección de una estructura TEDINFO.
G_BOXTEXT	22	Dirección de una estructura TEDINFO.
G_IMAGE	23	Dirección de una estructura BITBLK.
G_USERDEF	24	Dirección de una estructura USERBLK.
G_IBOX	25	BOXINFO, ver más abajo.
G_BUTTON	26	Dirección del string, que contiene el texto.
G_BOXCHAR	27	BOXINFO, ver más abajo.
G_STRING	28	Dirección del correspondiente string.
G_FTEXT	29	Dirección de una estructura TEDINFO.
G_FBOXTEXT	30	Dirección de una estructura TEDINFO.
G_ICON	31	Dirección de una estructura ICONBLK.
G_TITLE	32	Dirección del correspondiente string.

Para G_BOX, G_IBOX y G_BOXCHAR están contenidas en OB_SPEC informaciones, que se refieren a los caracteres, el marco y el color del correspondiente objeto. En este caso, solo en G_BOXCHAR están ocupados los 8 bits superiores y contienen allí el código ASCII del carácter que aparece en el box.

Contienen los siguientes valores para el marco:

```
      0 = ningún marco.
  1a 128 = El marco se encuentra 1 a 128 pixel dentro del correspon-
           diente objeto.
-1 a -128 = El marco se encuentra 1 a 128 pixel fuera del correspon-
           diente objeto.
```

Instalación de bits para el color del objeto: 1111 2222 3444 5555

```
1 Marco                 (0 a 15)
2 Texto                 (0 a 15)
3 Texto                 (0 = transparente, 1 = que cubre)
4 Diseño de relleno ( 0 a 7)
5 Color del interior del objeto
```

OB_FLAGS	Hex	Nro.de bit
SELECTABLE	&H0001	0
DEFAULT	&H0002	1
EXIT	&H0004	2
EDITABLE	&H0008	3
RBUTTON	&H0010	4
LASTOB	&H0020	5
TOUCHEXIT	&H0040	6
HIDETREE	&H0080	7
INDIRECT	&H0100	8

OB_STATE	Hex	Nro.de bit
SELECTED	&H0001	0
CROSSED	&H0002	1
CHECKED	&H0004	2
DISABLED	&H0008	3
OUTLINED	&H0010	4
SHADOWED	&H0020	5

Las estructuras analizadas en el párrafo anterior se activan en GFA-BASIC 3.0 con la siguiente sintaxis (leer y escribir):

```
OB_NEXT(tree%,obj&)
OB_HEAD(tree%,obj&)
OB_TAIL(tree%,obj&)

OB_TYPE(tree%,obj&)
OB_FLAGS(tree%,obj&)
OB_STATE(tree%,obj&)
OB_SPEC(tree%,obj&)

OB_X(tree%,obj&)
OB_Y(tree%,obj&)
OB_W(tree%,obj&)
OB_H(tree%,obj&)
```

En este caso tree% es la direción del árbol del objeto y obj& es el número del objeto, sobre el cual se solicita información.

La dirección del objeto se puede determinar además con

```
OB_ADR(tree%,obj&).
```

Estructura de información de texto (TEDINFO)

Offset	Contenido	Tipo	Significado
00	te_ptext	long	Señalador sobre el texto.
04	te_ptmplt	long	Señalador sobre máscara del texto.
08	te_pvalid	long	Señalador sobre el string para validez de los caracteres que se ingresan.
12	te_font	word	Oración de caracteres.
14	te_resvd	word	Reservado.
16	te_just	word	Alineamiento de texto
18	te_color	word	Color del box circundante.
20	te_resvd2	word	Reservado.
22	te_thickness	word	Espesor del marco.
24	te_txtlen	word	Longitud del texto.
26	te_tmplen	word	Longitud de la máscara del texto.

Estructura Icon (ICONBLK)

Offset	Contenido	Tipo	Significado
00	ib_pmask	long	Señalador sobre máscara Icon.
04	ib_pdata	long	Señalador sobre datos Icon.
08	ib_ptext	long	Señalador sobre texto Icon.
12	ib_char	word	Señalador dentro del Icon.
14	ib_xchar	word	Posición x del carácter.
16	ib_ychar	word	Posición y del carácter.
18	ib_xicon	word	Posición x del Icon.
20	ib_yicon	word	Posición y del Icon.
22	ib_wicon	word	Ancho del Icon.
24	ib_hicon	word	Altura del Icon.
26	ib_xtext	word	Posición x del texto.
28	ib_ytext	word	Posición y del texto.
30	ib_wtext	word	Ancho del texto en pixel.
32	ib_htext	word	Altura del texto en pixel.

34	ib_resvd	word	Reservado.

Estructura de Bit-image (BITBLK)

Offset	Contenido	Tipo	Significado
00	bi_pdata	long	Señalador sobre los datos del image.
04	bi_wb	word	Ancho del image en bytes.
06	bi_hl	word	Altura del image en pixel.
08	bi_x	word	Posición x del image.
10	bi_y	word	Posición y del image.
12	bi_color	word	Color del image.

Estructura del bloque de aplicación (USERBLK)

Offset	Contenido	Tipo	Significado
00	ub_code	long	Señalador sobre una rutina autodefinida para la representación del objeto.
04	ub_parm	long	Señalador sobre una estructura PARMBLK.

En este caso debe tratarse de rutinas assembler.

Estructura del bloque de parámetros (PARMBLK)

Offset	Contenido	Tipo	Significado
00	pb_tree	long	Señalador sobre el árbol del objeto.
04	pb_obj	word	Número del objeto.
06	pr_prevstate	word	Estado anterior.
08	pr_currstate	word	Estado actual.
10	pb_x	word	Posición x del objeto.
12	pb_y	word	Posición y del objeto.
14	pb_w	word	Ancho.
16	pb_h	word	Altura.
18	pb_xc	word	Posición x del rectángulo de clipping.
20	pb_yc	word	Posición y del rectángulo de clipping.
22	pb_wc	word	Ancho del rectángulo de clipping.
24	pb_hc	word	Altura del rectángulo de clipping.
28	pb_parm	word	Parámetros de la estructura USERBLK.

En las funciones AES ha de prestarse atención, que algunas funciones no solo dan un valor de función, sino también valores de retorno en variables puestas a disposición para tal fin. Otras funciones retornan valores reservados y se pueden solicitar a través de VOID o bien su abreviatura, el tilde "~". Cuando se indican direcciones se deben usar tipos de variables de por lo menos 4 bytes de longitud, y las indicaciones de las coordenadas se deben efectuar en tipos de variables de por lo menos 2 bytes de longitud.

Biblioteca de aplicación (application library)

Maneja los accesos a otras bibliotecas AES. El GFA-BASIC 3.0 llama automáticamente las funciones APPL_INIT y APPL_EXIT al iniciarse o

bien al abandonarse el programa .

APPL_INIT()

El programa actual se anuncia como aplicación GEM. La función APPL_INIT da el handle del programa actual de GFA-BASIC. El número de identificación es importante, por ej., para la instalación de fonts adicionales.

rueck Contiene el número de identificación (ap_id) de la aplicación (del programa) para el GEM. APPL_INIT se usa en lugar de APPL_READ, APPL_WRITE y MENU_REGISTER. Esta función es una función dummy en el GFA-BASIC, que no realiza un llamado del sistema operativo, ya que la APPL_INIT ya se ejecuta al iniciarse el intérprete GFA-BASIC. Se retorna el ap_id obtenido en este caso.

APPL_READ(id,len,adr_buffer)

id,len,adr_buffer: iexp

Con esta instrucción se pueden leer bytes de un buffer de eventos (message-buffer).

rueck	0 = Surgió un error.
id	Número de la aplicación, de cuyo buffer se lee.
len	Cantidad de bytes a leer.
adr_buffer	Dirección, a partir de la cual se han de depositar los bytes leídos.

APPL_WRITE(id,len,adr_buffer)

id,len,adr_buffer: iexp

De esta forma se escriben bytes en un buffer de eventos.

rueck	0 = Surgió un error.
id	Número de la aplicación, en cuyo buffer se escribe.
len	Número de bytes a escribir.
adr_buffer	Dirección, a la cual se dirigen los bytes.

APPL_FIND(fname$)

fname$: sexp

Busca el número de identificación de una aplicación, cuyo nombre de archivo es conocido, por ej. para el intercambio de información con otros programas que corren.

rueck Identificación del programa buscado, -1 = surgió un error, no se encontró el programa.

fname Contiene el nombre del archivo de 8 caracteres (sin sufijo) de la aplicación a buscar. El nombre del archivo debe tener

indispensablemente un largo de 8 caracteres y debe ser escrito en mayúscula, eventualmente se puede rellenar con espacios en blanco.

Ejemplo:

```
PRINT APPL_FIND("CONTROL")
```

--> Cuando el control-accesory está cargado da 2 (o algo parecido), sino da -1. Además de accesorys se puede buscar el programa propio "GFABASIC"(ap_id=0) y el screen-manager "SCRENMGR"(ap_id=1), que está a disposición por ej. para el handling de menús.

```
APPL_TPLAY(mem,num,scale)
APPL_TRECORD(mem,num)
```

mem,num,scale: iexp

Deben tener la capacidad de ejecutar y producir una serie de actividades del usuario (activar el ratón y teclas y la distancia entre ésto) del tipo de un grabador, tape play y record. Estas funciones no trabajan así como está indicado en la documentación GEM.

Con versiones de ROM más recientes estás funciones trabajan casi de acuerdo a lo indicado (en lugar de señalar 6 byte por evento se consumen 8 byte y el scale-factor también actúa diferente), pero como este comportamiento es todo menos confiable se aconseja no usar estas funciones. Por eso, tampoco se darán más descripciones.

```
APPL_EXIT()
```

APPL_EXIT quita un programa del AES. En este caso se libera el número de identificación del correspondiente programa, el cual queda a disposición de otros programas.

APPL_EXIT existe en GFA-BASIC solo como función dummy, ya que una instrucción QUIT/System ejecuta esto automáticamente.

rueck 0 = surgió un error

Biblioteca de eventos (event library)

Reacciona frente a ingresos a través del ratón, del teclado o de eventos dependientes del tiempo.

```
EVNT_KEYBD()
```

Espera un evento del teclado y retorna el correspondiente código del teclado.

rueck Código del teclado de la tecla pulsada (low byte) y código scan (high byte).

Ejemplo:

```
DO
  PRINT HEX$(EVNT_KEYBD(),4)
LOOP
```

--> Se visualiza este código del teclado en hexadecimal.

EVNT_BUTTON(clicks,mask,state[,mx,my,button,k_state])

clicks,mask,state: iexp
mx,my,button,k_state: ivar

Espera un evento del teclado del ratón a determinarse.

rueck	Cantidad de pulsaciones realizadas en el teclado del ratón.
clicks	Cantidad de pulsaciones necesarias del teclado del ratón.
mask	Máscara para la tecla deseada del ratón: Bit 0 = izquierda, bit 1 = derecha.
state	Estado deseado para desencadenar el evento. Instalación de los bits como en mask.
mx	Coordenada x al desencadenarse el evento a través del cursor del ratón.
my	Coordenada y al desencadenarse el evento a través del cursor del ratón.
button	Estado de las teclas del ratón, igual que en state.
k_state	Estado de las teclas de conmutación del teclado al desencadenarse el evento: Bit 0 = Tecla Shift derecha Bit 1 = Tecla Shift izquierda Bit 2 = Tecla Control Bit 3 = Tecla Alternate

Los parámetros mx,my,button y k_state son opcionales, estos valores también se pueden solicitar a través de GINTOUT(1) a GINTOUT(4).

Ejemplo:

```
DO
  SELECT EVNT_BUTTON(2,1,1,mx%,my%,bu%,kb%)
  CASE1
    PLOT mx%,my%
  CASE2
    PBOX mx%-10,my%-10,mx%+10,my%+10
  ENDSELECT
LOOP UNTIL BTST(kb%,2)
```

--> Espera clicks del ratón con la tecla izquierda del ratón. Cuando se trata de un click simple se fija un punto, cuando se trata de un click doble se fija un cuadrado. Este ejemplo se finaliza, si además del click se pulsa la tecla Control.

EVNT_MOUSE(flags,mx,my,mw,mh,mcur_x,mcur_y,button,k_state)

flags,mx,my,mw,mh: iexp
mcur_x,mcur_y,button,k_state: ivar

Espera a que el señalador del ratón entre o salga de una porción rectangular de la pantalla.

rueck	Reservado, siempre 1.
flags	Anuncia la entrada (0) o la salida (1) de la porción rectangular de la pantalla.
mx	Coordenada x izquierda.
my	Coordenada y superior.
mw	Ancho.
mh	Altura de la porción rectangular de la pantalla.
mcur_x	Coordenada x.
mcur_y	Coordenada y del cursor del ratón.
button	Estado de las teclas del ratón al iniciarse el evento: Bit 0 = Tecla derecha del ratón. Bit 1 = Tecla izquierda del ratón.
k_state	Estado del teclado al iniciarse el evento: Bit 0 = Tecla Shift derecha. Bit 1 = Tecla Shift izquierda. Bit 2 = Tecla Control. Bit 3 = Tecla Alternate.

Ejemplo:

```
DO
  EVNT_MOUSE(0,100,100,200,90,mx%,my%,bu%,kb%)
  IF bu%
    PBOX mx%-10,my%-10,mx%+10,my%+10
  ELSE
    PLOT mx%,my%
  ENDIF
LOOP UNTIL BTST(kb%,2)
```

--> Se espera hasta que el cursor del ratón se encuentra en un rectángulo. Si se pulsa la tecla del ratón, se dibuja un rectángulo, sino un punto. Se finaliza pulsando la tecla Control, cuando el señalador está dentro de un rectángulo.

Atención: Aunque durante la ejecución de rutinas GM se puede pulsar Control-Alternate, recién tendrá efecto después de retornar al BASIC.

Así, por ej. ~EVNT_TIMER(3600000) o DELAY 3600 intercalan una hora de pausa, hasta que el calculador retorna del GEM.

EVNT_MESAG(adr_buffer)

adr_buffer: iexp

Espera que aparezca un mensaje (message) en el buffer de eventos.

rueck	Reservado, siempre 1.
adr_buffer	Dirección del buffer de 16 byte de longitud (message buffer) para captar el mensaje (comparar MENU(1) a MENU(8)). Cuando para adr_buffer se indica 0 se usa independientemente el message buffer propio del GFA-BASIC, o sea MENU(1) a MENU(8).

```
EVNT_TIMER(count)
```

count: iexp

Espera un lapso que se debe indicar en milisegundos (comparar DELAY).

rueck	Reservado, siempre 1.
count	Cantidad de milisegundos.

```
EVNT_MULTI(flags,clicks,mask,state,m1_flags,m1_x,m1_y,m1_w,m1_h,
          m2_flags,m2_x,m2_y,m2_w,m2_h,adr_buffer,count
          [,mcur_x,mcur_y,button,k_state,key,anz_clicks])
```

```
flags,clicks,mask,state,m1_flags,m1_x,m1_y,m1_w,m1_h: iexp
m2_x,m2_y,m2_w,m2_h,adr_buffer,count: iexp
mcur_x,mcur_y,button,k_state,key,anz_clicks: ivar
```

Espera que se inicien varios eventos.

rueck	Contiene el evento que realmente ha tenido lugar.La instalación de bits es igual que en ev_mflags.
flags	Determina a qué evento debe esperar la aplicación. En este caso las equivalencias son:

Bit 0	Teclado	MU_KEYBD
Bit 1	Botón del ratón	MU_BUTTON
Bit 2	Primer evento del ratón	MU_M1
Bit 3	Segundo evento del ratón	MU_M2
Bit 4	Evento de mensaje	MU_MESAG
Bit 5	Timer	MU_TIMER

anz_clicks	Cantidad de pulsaciones activadas del teclado del ratón.

Los siguientes parámetros ya se han analizado en EVNT_MOUSE, EVNT_KEYBD, EVNT_BUTTON y EVNT_MESAG. Sin embargo, ha de prestarse atención, que se pueden esperar dos diferentes eventos del ratón (m1 y m2). En ON MENU, que usa esta rutina internamente, se ajustan los parámetros a través de ON MENU xxxGOSUB o bien se indica count directamente en la instrucción.

MENU(1) a MENU(8)	Message buffer
MENU(9)	Valor de retorno
MENU(10) = mcur_x	Posición del cursor del ratón
MENU(11) = mcur_y	Posición del cursor del ratón
MENU(12) = button	Botón del ratón
MENU(13) = k_state	BIOS(11,-1)
MENU(14) = key	Código Scan y ASCII
MENU(15) = anz_clicks	Cantidad de clicks del ratón

```
EVNT_DCLICK(new,get_set)
```

new,get_set: iexp

Fija la velocidad para clicks dobles del ratón.

rueck	Velocidad anterior
new	Velocidad nueva

```
get_set   Tipo de ajuste:
           0= obtener el valor actual, sin prestar atención a new.
           1 = fijar el nuevo valor para new.
```

Biblioteca de menús (menu library)

Asume la función de dibujar una zona de menú y su administración

```
MENU_BAR(tree,flag)

tree,flag:  iexp
```

Conecta o desconecta una zona de menú (proveniente de un resource-file).

```
Compare: MENU x$() y MENU KILL.

rueck    0 = Surgió un error.
tree     Dirección del árbol del objeto del menú.
flag     1 = Conectar la zona de menú.
         0 = Desconectar la zona del menú (siempre hacia el final del
         programa).

MENU_ICHECK(tree,item,flag)

tree,item,flag:  iexp
```

Fija o borra un tilde delante de una inscripción del menú (por eso deberían existir siempre dos espacios en blanco delante de cada inscripción de menú).

```
Compare:  MENU x,0 y MENU x,1.

rueck    0 = Surgió un error.
tree     Dirección del árbol del objeto del menú.
item     Número del objeto de la correspondiente inscripción del menú.
flag     0 = Fijar un tilde.
         1 = Borrar un tilde.

MENU_IENABLE(tree,item,flag)

tree,item,flag:  iexp
```

Produce la conexión y desconexión de inscripciones del menú. La correspondiente inscripción de menú se representa entonces en letra de color gris claro.

```
Comparar: MENU x,2 y MENU x,3.

rueck    0 = Surgió un error.
tree     Dirección del árbol del menú.
item     Número del objeto de la correspondiente inscripción de menú.
flag     0 = Desconectar.
         1 = Conectar.
```

MENU_TNORMAL(tree,title,flag)

tree,title,flag: iexp

Conmuta títulos de menú entre representación invertida y normal.

Compare: MENU OFF.

rueck	0 = Surgió un error.
tree	Dirección del árbol del objeto de menú.
title	Número del objeto del correspondiente título del menú.
flag	0 = Representación invertida. 1 = Representación normal.

MENU_TEXT(tree,item,new_text$)

tree,item: iexp
new_text$: sexp

Modifica el texto de una inscripción de menú. Esta función permite adaptar inscripciones de menú al estado actual del programa.

rueck	0, cuando surge un error.
tree	Dirección del árbol del objeto del menú.
item	Número de la inscripción del objeto que se quiere modificar.
new_text$	String que contiene la nueva inscripción de menú (no debe superar la longitud de la inscripción anterior).

MENU_REGISTER(ap_id,m_text$)

ap_id: iexp
m_text$: sexp

Inscribe los nombres de accesorios bajo el primer título del menú. En este caso se pueden instalar como máximo 6 inscripciones de menú accesory.

Sin embargo, esta función solo puede ser aplicada convenientemente con un accesory (solo en el programa compilado).

rueck	Caracterización para la inscripción del menú del correspondiente accesory: 0 a 5 para la primera a sexta inscripción accesory. -1, si no es posible otra inscripción.
ap_id	Número de identificación del correspondiente accesory.
m_text$	Dirección de la correspondiente inscripción de menú.

Ejemplo: Se suprime en este caso, ya que solo es posible en accesorios.

Biblioteca de objetos (object library)

Contiene rutinas para la definición, representación y modificación de objetos.

```
OBJ_ADD(tree,parent,child)

tree,parent,child:  iexp
```

Agrega un objeto a un determinado árbol del objeto y establece la conexión entre los objetos existentes y el nuevo objeto.

```
rueck      0 = Surgió un error.
tree_adr   Dirección del correspondiente árbol del objeto.
parent     Número de objeto del "objeto parental", al cual se agrega el
           nuevo objeto.
child      Número de objeto del "objeto hijo", que será acoplado al
           "objeto parental".
```

```
OBJC_DELETE(tree,del_obj)

tree,del_obj:  iexp
```

Modifica los señaladores next, head y tail de un objeto dentro del correspondiente árbol del objeto, de tal manera que no se puede solicitar más hasta que este señalador vuelva a ser restaurado.

```
rueck      0 = Surgió un error.
tree       Dirección del correspondiente árbol del objeto.
del_obj    Número de objeto del objeto a borrar.
```

```
OBJ_DRAW(tree,start_obj,depth,cx,cy,cw,ch)

tree,star_obj,depth,cx,cy,cw,ch:  iexp
```

Esta función representa en la pantalla objetos completos o sus partes. Opcionalmente se pueden indicar las coordenadas de una sección rectangular de la pantalla, al cual se limitará la representación.

```
rueck      0 = Surgió un error.
tree       Dirección del correspondiente árbol del objeto.
start_obj  Número de objeto del primer objeto a representar.
start_obj  Número de objeto del primer objeto a representar.
            0 = Dibujar solo el primer objeto.
depth      Cantidad de planos del objeto a representar.
cx         Coordenada x izquierda.
cy         Coordenada y superior.
cw         Ancho y
ch         Altura del rectángulo de límite.
```

```
OBJC_FIND(tree,start_obj,depth,fx,fy)

tree,start_obj,depth:  iexp
fx,fy:  ivar
```

Esta función hace posible indicar las coordenadas del cursor del ratón y da el número del objeto que se encuentra en la posición indicada de la pantalla.

```
rueck      Número del objeto hallado.
            -1 = no se halló ningún número de objeto.
tree       Dirección del árbol de objeto a analizar.
```

```
start_obj   Número del primer objeto a analizar.
depth       Cantidad de planos del objeto a analizar.
             0 = solo el "objeto madre".
fx          Coordenada x
fy          Coordenada y de la posición de pantalla a indicar.
```

```
OBJC_OFFSET(tree,obj,x_abs,y_abs)

tree,obj:  iexp
x_abs,y_abs:  ivar
```

Calcula las coordenadas absolutas de la pantalla de un objeto a indicar.

```
rueck     0 = Surgió un error.
tree      Dirección del correspondiente árbol del objeto.
obj       Número del objeto a indicar.
x_abs     Las coordenadas x absolutas calculadas y
y_abs     las coordenadas y absolutas calculadas.
```

```
OBJC_ORDER(tree,obj,new_pos)

tree,obj,new_pos:  iexp
```

Modifica dentro de los planos del objeto la secuencia de un objeto a indicar.

```
rueck     0 = Error.
tree      Dirección del correspondiente árbol del objeto.
obj       Número del nuevo plano del objeto:
           -1 = un plano más arriba
            0 = plano inferior
            1 = plano inferior + 1
            2 = plano inferior + 2
                etc.
```

```
OBJC_EDIT(tree,char,old_pos,flag,new_pos)

tree,obj,char,old_pos,flag:  iexp
new_pos:  ivar
```

Hace posible el ingreso y la edición de textos en los tipos del objeto G_TEXT y G_BOXTEXT.

```
rueck       0 = Surgió un error.
tree        Dirección del correspondiente árbol del objeto.
obj         Número del correspondiente objeto .
char        Carácter ingresado (inclusive código Scan).
old_pos     Posición del carácter den el string de ingreso.
flag        Selección de funciones:
            0 ED_START = Reservado - llamado dummy.
            1 ED_INIT  = Calcular string formateado y conectar el
                         cursor.
            2 ED_CHAR  = Elaborar el correspondiente carácter y
                         volver a indicar el string.
            3 ED_END   = Desconectar el cursor.
new_pos     Nueva posición del carácter en el string de ingreso.
```

```
OBJC_CHANGE(tree,obj,res,cx,cy,cw,ch,new_status,re_draw)

tree,obj,res,cx,cy,cw,ch,new_status.re_draw:  iexp
```

Esta función modifica el estado de un objeto (OB_STATE) e invierte eventualmente aquella porción del objeto que se encuentra dentro de la zona de clip. Normalmente se modifica directamente OB_STATE y luego se llama OBJC_DRAW.

rueck	0 = Error.
tree	Dirección del correspondiente árbol del objeto.
obj	Número del objeto a modificar.
res	Reservado, siempre 0.
cx	Coordenada x izquierda.
cy	Coordenada y superior.
cw	Ancho y
ch	Altura del rectángulo de límite.
new_status	Nuevo estado del objeto (ver OB_STATE).
re_draw	0 = No dibujar de nuevo el objeto. 1 = Dibujar de nuevo el objeto.

Biblioteca de formularios (form library)

Contiene rutinas para la administración de formularios.

```
FORM_DO(tree,start_obj)

tree,start_obj:  iexp
```

Esta función asume la administración completa de un objeto de formularios, hasta que se activa un objeto con estado EXIT o TOUCH_EXIT.

rueck	Número del objeto, que ha sido activado para la finalización (cuando es un click doble se fija el 15-avo bit).
tree	Dirección del correspondiente árbol del objeto.
start_obj	Número del primer objeto en el árbol del objeto.

```
FORM_DIAL(flag,mi_x,mi_y,mi_w,mi_h,ma_x,ma_y,ma_w,ma_h)

flag,mi_x,mi_y,mi_w,mi_h,ma_x,ma_y,ma_w,ma_h:  iexp
```

Esta función sirve para reservar o liberar secciones rectangulares de la pantalla y para dibujar rectángulos que se comprimen o que se expanden.

rueck	0, cuando surgen errores.	
flag	Tipo de función:	
	0 = FMD_START	Reserva una zona de la pantalla.
	1 = FMD_GROW	Dibuja un rectángulo que se expande .
	2 = FMD_SHRINK	Dibuja un rectángulo que se comprime.
	3 = FMD_FINISH	Vuelve a liberar una zona reservada de la pantalla.

mi_x Coordenadas del rectángulo
mi_y en su mínima expresión.
mi_w
mi_h

ma_x Coordenadas del rectángulo
ma_y en su máxima extensión.
ma_w
ma_h

Explicación: FORM_DIAL(1,0,0,0,0,100,100,300,100) dibuja un rectángulo que se expande 0,0,0,0 significa, que el rectángulo aparece del medio del rectángulo de destino.

FORM_ALERT(button,string$)

button: iexp
string$: sexp

Indica un mensaje general de alerta (ALERT_BOX).

rueck Número del botón con el que abandonó el Alert_box.
button Número del botón default (por defecto).
0 = ninguno
1 = el primero
2 = el segundo
3 = el tercero
string$ Dirección del string que contiene el mensaje.
El string tiene el siguiente formato:

[i][mensaje][button]

en este caso son:

i exactamente un número 0 = ningún icon
1 = signo de exclamación
2 = signo de pregunta
3 = cartel stop.
mensaje a lo sumo 5 líneas de texto con cada vez 30 caracteres , separados entre sí con I.
button a lo sumo 3 nombres de button separados con I.

Ejemplo:

```
~FORM_ALERT(1,ERR$(100))
```

FORM_ERROR(err)

err: iexp

Indica un mensaje de alerta para errores MS-DOS.

rueck Número del button, con el que se abandonó el mensaje de alerta .
err Número de error MS-DOS.

Form_CENTER(tree,fx,fy,fw,fh)

tree: iexp

fx,fy,fw,fh: ivar

Estas funciones centran las coordenadas que se han de transferir.

rueck	Reservado, siempre 1.
tree	Dirección del correspondiente árbol del objeto.
fx	Coordenada x izquierda.
fy	Coordenada y superior.
fw	Ancho y
fh	Altura del rectángulo centrado.

FORM_KEYBD(tree,obj,next_obj,char,new_obj,next_char)

tree,obj,next_obj,char,new_obj: iexp
next_char: ivar

Posibilita el ingreso de datos a través del teclado en un formulario (ver OBJ_EDIT).

rueck	0 = Objeto pulsado con estado EXIT. Mayor que 0 = Diálogo no finalizado aún.
tree	Dirección del correspondiente árbol del objeto.
obj	Número del objeto EDIT actual.
next_obj	Número del siguiente objeto EDIT.
char	Carácter leído e incorporado.
new_obj	Nuevo objeto EDIT para el siguiente llamado.
next_char	Siguiente carácter.

Aclaración: Esta rutina es una subrutina de FORM_DO.

FORM_BUTTON(tree,obj,clicks,new_obj)

tree,obj,clicks: iexp
new_obj: ivar

Hace posible ingresar datos del ratón en un formulario.

rueck	0 = Al final se activó un objeto con estado EXIT. Mayor que 0 = El diálogo no ha sido finalizado aún.
tree	Dirección del correspondiente árbol del objeto.
obj	Objeto actual.
clicks	Cantidad de pulsaciones del ratón.
new_obj	Nuevo objeto actual.

Aclaración: Esta rutina es una subrutina de FORM_DO.

Biblioteca de gráficos (graphics librark)

GRAF_RUBBERBOX(rx,ry,min_w,min_h[last_w,last_h]

rx,ry,min_w,min_h: iexp

```
last_w,last_h:  ivar
```

Con la tecla del ratón pulsada la función GRAF_RUBBERBOX representa en la pantalla el contorno de un rectángulo. La coordenada x izquierda y la coordenada y superior son fijas en este caso, pero el canto enfrentado en diagonal sal como el ancho y la altura del rectángulo se modifican de acuerdo con la posición del cursor del ratón. El llamado de la función debería ocurrir únicamente con la tecla del ratón pulsada, ya que la función se interrumpe cuando se deja de pulsar la tecla del ratón.

rueck	0 = Srugió un error.
rx	Coordenada x izquierda.
ry	Coordenada y superior.
min_w	Ancho menor y
min_h	Altura del rectángulo.
last_w	Ancho y
last_h	Altura del rectángulo cuando se interrumpe la función.

Ejemplo:

```
DO
  ~EVNT_BUTTON(1,1,1,mx%,my%,bu%,kb%)
  ~GRAF_RUBBERBOX(mx%,my%,1,1,w%,h%)
  BOX mx%,my%,mx%+w%,my%+h%
LOOP UNTIL BTST(kb%,3)
```

--> Espera el click del ratón. Entonces se esboza un rectángulo (elástico) que se dibuja a continuación. Se interrumpe con Alternate y la tecla izquierda del ratón.

```
GRAF_DRAG_BOX(iw,ih,ix,iy,rx,ry,rw,rh[last_ix,last_iy]
```

```
iw,ih,ix,iy,rx,ry,rw,rh:  iexp
last_ix,last_iy:  ivar
```

Esta función sirve para desplazar un rectángulo (interno) dentro de otro rectángulo mayor y fijo (rectángulo de marco). La función se debería llamar únicamente con la tecla del ratón pulsada, ya que la función se interrumpe cuando se deja de pulsar una tecla del ratón.

rueck	0 = Surgió un error.
iw	Ancho y
ih	Altura del rectángulo interno.
ix	Coordenada x izquierda y
iy	Coordenada y superior del rectángulo interno cuando se llama la función.
rx	Coordenada x izquierda.
ry	Coordenada y superior.
rw	Ancho y
rh	Altura del rectángulo de marco.
last_ix	Coordenada x izquierda y
last_iy	Coordenada y superior del rectángulo interno cuando se interrumpe el programa.

GRAF_MOVEBOX(w,h,sx,sy,dx,dy)

w,h,sx,sy,dx,dy: iexp

La función GRAF_MOVEBOX dibuja un rectángulo desplazable con ancho y altura constantes.

rueck	0 = Surgió unerror.
w	Ancho del rectángulo desplazable.
h	Altura del rectángulo desplazable.
sx	Coordenada x izquierda del rectángulo cuando se llama la función.
sy	Coordenada y superior del rectángulo cuando se llama la función.
dx	Coordenada x izquierda del rectángulo cuando se llama la función.
dy	Coordenada y superior del rectángulo cuando se llama la función.

Ejemplo:

```
~GRAF_MOVEBOX(10,20,200,20,80,120)
```

GRAF_GROWBOX(sx,sy,sw,sh,dx,dy,dw,dh)

sx,sy,sw,sh,dx,dy,dw,dh: iexp

rueck	0 = Surgió un error.
sx	Coordenada x izquierda.
sy	Coordenada y superior.
sw	Ancho del rectángulo al comenzar la función.
sh	Altura del rectángulo al comenzar la función.
dx	Coordenada x izquierda.
dy	Coordenada y superior.
dw	Ancho del rectángulo al finalizar la función.
dh	Altura del rectángulo al finalizar la función

Ejemplo:

```
~GRAF_GROWBOX(100,100,10,10,0,0,300,180)
```

GRAF_SHRINKBOX(sx,sy,sw,sh,dx,dy,dw,dh)

sx,sy,sw,sh,dx,dy,dw,dh: iexp

Esta función dibuja un rectángulo que se comprime.

rueck	0 = Surgió un error.
sx	Coordenada x izquierda.
sy	Coordenada y superior.
sw	Ancho del rectángulo al finalizar la función.
sh	Altura del rectángulo al finalizar lafunción.
dx	Coordenada x izquierda.
dy	Coordenada y superior.
dw	Ancho del rectángulo al comenzar la función.
dh	altura del rectángulo al comenzar la función.

Ejemplo:

```
~GRAF_SHRINKBOX(0,0,300,180,100,100,10,10)
```

GRAF_WATCHBOX(tree,obj,in_state,out_state)

tree,obj,in_state,out_state: iexp

Esta función supervisa un objeto (BOX) en un árbol de resource mientras se pulsa una tecla del ratón. El estado del objeto se modifica cada vez que el señalador del ratón alcanza o abandona la zona correspondiente (normal selected/normal).

rueck	Indica el estado del cursor del ratón cuando se deja de pulsar la tecla del ratón, 0 = fuera y 1 = dentro del objeto a supervisar.
tree	Dirección del correspondiente árbol del objeto.
obj	Número del objeto a supervisar.
in_state	Estado (OB_STATE) del objeto a supervisar, cuando el cursor del ratón se encuentra dentro del objeto.
out_state	Estado del objeto a supervisar (OB_STATE), cuando el cursor del ratón se encuentra fuera del objeto. La correspondiente instalación de bits se encuentra bajo OB_STATE.

GRAF_SLIDEBOX(tree,parent_obj,slider_obj,flag)

tree,parent_obj,slider_obj,flag)

Esta función hace posible solicitar los llamados de posicionadores de ventana. En forma similar a lo que ocurre con DRAG_BOX se puede desplazar en este caso un rectángulo (posicionador) dentro de un rectángulo de marco, aunque el rectángulo desplazable sólo se puede mover en forma horizontal o bien perpendicular. El rectángulo desplazable debe ser además un "child object" del rectángulo de marco dentro del árbol del objeto. La función se debería llamar únicamente pulsando una tecla del ratón, ya que la función se interrumpe cuando se deja de pulsar la tecla del ratón.

rueck	Posición relativa del posicionador dentro del rectángulo de marco: 0 = Izquierda o bien arriba. 1000 = Derecha o bien abajo.
tree	Dirección del correspondiente árbol del objeto.
parent_obj	Número de objeto del "Rectángulo de marco".
slider_obj	Número de objeto del posicionador.
flag	Dirección de desplazamiento: 0 = Horizontal 1 = Perpendicular.

GRAF_HANDLE(char_w,char_h,box_h)

char_w,char_h,box_w,box_h): ivar

Da la caracterización de la workstation VDI, que se usa internamente para los llamados de AES, y la extensión de un carácter de la

simbología estándar.

rueck	Caracterización de la workstation actual.
char_w	Ancho de un carácter de la simbología estándar en pixel.
char_h	Altura de un carácter de la simbología estándar en pixel.
box_w	Ancho de un rectángulo que comprende un carácter estándar.
box_h	Altura de un rectángulo, que comprende un carácter estándar.

GRAF_MOUSE(m_form,diseño_adr)

m_form,diseño_adr: iexp

Esta función posibilita la elección de la forma del cursor del ratón. Para eso, se dispone de ocho formas predefinidas o una forma definida por el usuario. Sin embargo, resulta más confortable emplear la instrucción DEFMOUSE.

rueck 0 = Surgió un error.

m_form Número de la forma del cursor del ratón:

0 =	ARROW	Flecha
1 =	TEXT_CRSR	Línea perpendicular
2 =	HOURGLASS	Abeja
3 =	POINT HAND	Mano señalando
4 =	FLAT_HAND	Mano expandida
5 =	THIN_CROSS	Cruz fina
6 =	THICK_CROSS	Cruz gruesa
7 =	OUTL_CROSS	Contorno de la cruz
255 =	USER_DEF	Define el usuario
256 =	M_OFF	Desconectar el cursor del ratón.
257 =	M_ON	Conectar el cursor del ratón.

diseño_adr Dirección, a partir de la cual se encuentran las informaciones de bits para la forma autodefinida del cursor del ratón. En este caso se esperan 37 words de acuerdo con lo siguiente:

1 =	Coordenada x y
2 =	Coordenada y del punto de acción.
3 =	Cantidad de planos color, siempre 1.
4 =	Color de la máscara, siempre 0.
5 =	Color del cursor del ratón, siempre 0.
6 a 21 =	Forma de la máscara.
22 a 37 =	Forma del cursor del ratón.

GRAF_MKSTATE(mx,my,m_state,k_state)

mx,my,m_state,k_state: iexp

Esta función da las coordenadas del cursor del ratón así como el estado de las teclas del ratón y del teclado.

Esta es una rutina AES para solicitar el ratón. Contrariamente a MOUSEX etc. el programa se detiene, cuando la flecha figura en una

línea del menú.

rueck	Reservado, siempre 1.
mx	Coordenada x actual y
my	Coordenada y del cursor del ratón.
m_state	Estado del teclado del ratón: Bit 0 = Tecla izquierda del ratón. Bit 1 = Tecla derecha del ratón.
k_state	Estado del teclado con la siguiente instalación de bits: (Bit fijado = Tecla pulsada) 1 = Tecla Shift derecha 2 = Tecla Shift izquierda 4 = Tecla Control 8 = Tecla Alternate

Biblioteca Scrap (scrap library)

Contiene rutinas, que hacen posible intercambiar datos entre diferentes aplicaciones.

SCRP_READ(camino$)

camino$: svar

Esta función lee del buffer de textos, que sirve para la comunicación de programas GEM ejecutados secuencialmente. Estó se diseñó especialmente para ubicar nombres de ordenadores y archivos.

rueck	0 = Surgió un error.
camino$	Camino de acceso para el clipboard.

SCRP_WRITE(camino$)

camino$: sexp

Esta función escribe algo en el scrap-directory.

rueck	0 = Surgió un error.
camino$	Nuevo camino de acceso

Ejemplo:

```
~SCRP_WRITE("C:\TMP\A.X")
...
...
...
~SCRP_READ(a$)
PRINT a$
```

Biblioteca selectora de archivos (file selector library)

Hace posible seleccionar un archivo de un índice señalado o directamente indicando el camino de acceso.

FSEL_INPUT(path$,name$,[button])

path$,name$: svar
button: ivar

Esta función llama al box selector de archivos estándar. Equivale a la instrucción Fileselector. En ese caso se arma en forma independiente la designación completa del archivo a partir de los nombres del camino y del archivo. Eso se puede hacer aquí con RINSTR(path$,"\").

rueck	0 = Surgió un error.
path$	Camino de acceso conectado previamente; después de la detención de la función contiene el camino de acceso indicado por el usuario.
name$	Nombre preseleccionado del archivo; después de la detención de la función contiene el nombre del archivo indicado por el usuario.
button	Estado del button: 0 = Se seleccionó el button de "Detención" 1 = Se seleccionó el button "Ok".

Biblioteca de ventanas (window library)

Asume la administración de ventanas.

WIND_CREATE(attr,wx,wy,ww,wh)

attr,wx,wy,ww,wh: iexp

Esta función anuncia una nueva ventana y fija los atributos de la ventana y su tamaño máximo. Esta función retorna la caracterización actual de la ventana (window handle).

Comparar: OPENW#n,x,y,w,h,attr

rueck 0 = Surgió un error, sino da la caracterización de la ventana.

attr Atributos de la ventana con la siguiente instalación:

			Bit-no
&H0001	NAME	Título con nombre.	0
&H0002	CLOSE	Campo de cierre.	1
&H0004	FULL	Campo para tamaño completo.	2
&H0008	MOVE	Barra de desplazamiento.	3
&H0010	INFO	Línea Info.	4
&H0020	SIZE	Ajuste del tamaño de ventana.	5
&H0040	UPARROW	Flecha hacia arriba.	6
&H0080	DNARROW	Flecha hacia abajo.	7
&H0100	VSLIDE	Posicionador vertical	8
&H0200	LFARROW	Flecha hacia la izquierda.	9
&H0400	RTARROW	Flecha hacia la derecha.	10
&H0800	HSLIDE	Posicionador horizontal	11

wx	Coordenada x izquierda máxima.
wy	Coordenada y superior.
ww	Ancho de la ventana.
wh	Altura de la ventana.

WIND_OPEN(handle,wx,wy,ww,wh)

handle,wx,wy,ww,wh: iexp

Esta función representa en la pantalla una ventana generada previamente con WIND_CREATE.

Comparar: OPENW

rueck 0 = Apareció una ventana.
handle Caracterización de la ventana.
wx Coordenada x izquierda.
wy Coordenada y superior.
ww Ancho y
wh Altura de la ventana al comenzar la función.

WIND_CLOSE(handle)

handle: iexp

Esta función representa lo opuesto a WIND_OPEN y cierra la correspondiente ventana.

Comparar: CLOSEWw

rueck 0 = Surgió un error.
handle caracterización de la correspondiente ventana.

WIND_DELETE(handle)

handle: iexp

Esta función borra una ventana y libera el espacio de memoria reservado así como la correspondiente caracterización de la ventana.

Comparar: CLOSEW

rueck 0 = Surgió un error
handle Caracterización de la correspondiente ventana.

WIND_GET(handle,field,w1,w2,w3,w4)

handle,field: iexp
w1,w2,w3,w4: ivar

Esta función da informaciones sobre una ventana según el número de función.

rueck 0 = Surgió un error
handle Caracterización de la correspondiente ventana.

field De acuerdo con el número de función se retornan informaciones sobre la correspondiente ventana en wi_gw1, wi_gw2, wi_gw3 y wi_gw4.

4 WX_WORKXYWH calcula las coordenadas de la zona de trabajo:
- w1 Coordenada x izquierda.
- w2 Coordenada y superior.
- w3 Ancho y
- w4 Altura de la correspondiente ventana.

5 WF_CURRXYWH calcula las coordenadas de toda la ventana inclusive los márgenes:
- w1 Coordenada x izquierda.
- w2 Coordenada y superior.
- w3 Ancho y
- w4 Altura de la correspondiente ventana.

6 WF_PREVXYWH calcula el tamaño total de la ventana anterior:
- w1 Coordenada x izquierda
- w2 Coordenada y superior.
- w3 Ancho y
- w4 Altura de la correspondiente ventana.

7 WF_FULLXYWH calcula el tamaño total de la ventana su máxima extensión posible:
- w1 Coordenada x izquierda.
- w2 Coordenada y superior.
- w3 Ancho y
- w4 Altura de la correspondiente ventana.

8 WF_HSLIDE da la ubicación del posicionador:
- w1 (1 = bien a la izquierda, 1000 = bien a la derecha)

9 WF_VSLIDE indica la ubicación del posicionador perpendicular:
- w1 (1 = bien arriba, 1000= bien abajo)

10 WF_TOP da la caracterización de la ventana superior:
- w1 Caracterización de la ventana

11 WF_FIRSTXYWH indica las coordenadas del primer rectángulo dentro de la lista de rectángulos de la correspondiente ventana:
- w1 Coordenada x izquierda.
- w2 Coordenada y superior.
- w3 Ancho de la correspondiente ventana.
- w4 Altura de la correspondiente ventana.

12 WF_NEXTXYWH indica las coordenadas del siguiente rectángulo dentro de la lista de rectángulos de la correspondiente ventana:
- w1 Coordenada x izquierda.
- w2 Coordenada y superior.
- w3 Ancho de la correspondiente ventana.
- w4 Altura de la correspondiente ventana.

13 WF_RESVD está reservado.

15 WF_HSLIZE calcula el tamaño del posicionador horizontal con respecto a la barra de desplazamiento:

```
        w1      -1 = Tamaño mínimo
                   (1 = pequeño,  1000 = Ancho máximo)

    16  WF_VSLIZE calcula el tamaño del posicionador
        perpendicular con respecto a la barra de
        desplazamiento:
         w1     -1 = Tamaño mínimo
                   (1 = pequeño, 1000 = Ancho máximo)
```

```
WIND_SET(handle,field,w1,w2,w3,w4)

handle,field:  iexp
w1,w2,w3,w4:   ivar
```

Esta función modifica de acuerdo con el número de función partes de la correspondiente ventana.

```
rueck     0 = Surgió un error.
handle    Caracterización de la correspondiente ventana.

field     De acuerdo con el número de función se modifican
          partes de la corerpondiente ventana:

          1   WF_KIND   Fija nuevas partes de la ventana (fijado
              igual que en WIND_CREATE).
                w1  Nueva parte de la ventana.

          2   WF_NAME fija un nuevo título de ventana:
                w1 y
                w2 contienen la dirección del string.

          3   WF_INFO fija una nueva línea Info:
                w1 y
                w2 contienen la dirección del string.

          5   WF_CURRXYWH fija el tamaño de la ventana:
                w1  Coordenada x izquierda.
                w2  Coordenada y superior.
                w3  Ancho de la correspondiente ventana.
                w4  Altura de la correspondiente ventana.

          8   WF_HSLIDE ubica el posicionador horizontal:
                w1  (1 = bien a la izquierda, 1000 = bien a la
                     derecha)

          9   WF_VSLIDE ubica el posicionador vertical:
                w1  (1 = bien arriba,  1000 = bien abajo)

         10   WF_TOP fija la ventana (actual) superior.

         14   WF_NEWDESK fija el nuevo árbol de menú desktop:
                w1   High-Byte.
                w2   Low-Byte de la dirección del árbol de menú.
                w3   Número del primer objeto a dibujar.

         15   WF_HSLIZE fija el tamaño del posicionador horizontal
              con respecto a la barra de desplazamiento:
```

```
                wl    -1 = Tamaño mínimo
                         (1 = pequeño, 1000 = Ancho máximo)

          16    WF_VSLIZE fija el tamaño del posicionador vertical con
                respecto a la barra de desplazamiento:
                wl    -1 = Tamaño mínimo
                         (1 = pequeño, 1000 = tamaño máximo)
```

```
WIND_FIND(fx,fy)

fx,fy:  iexp
```

Esta función obtiene la caracterización de una ventana en las coordenadas a indicar.

```
rueck     0 = Ninguna ventana con estas coordenadas, sino carac-
              terización de la ventana hallada.
fx        Coordenada x de la correspondiente posición en la pantalla.
fy        Coordenada y en la correspondiente posición en la pantalla.
```

```
WIND_UPDATE(flag)

flag:  iexp
```

Esta función coordina todas las medidas que están relacionadas con el armado de la pantalla, especialmente en relación con menús Pull-Down.

```
rueck     0 = Surgió un error.
flag      Número de función:
           0 = END_UPDATE   Se cerró el armado de la pantalla.
           1 = BEG_UPDATE   Comienza el armado de la pantalla.
           2 = END_MCTRL    Aplicación entrega el control del ratón.
           3 = BEG_MCTRL    Aplicación asume todo el control del
               ratón, en este caso están cerrados especialmente los
               menús Pull-Down.
```

```
WIND_CALC(w_type,attr,ix,iy,iw,ih,ox,oy,ow,oh)

w_type,attr,ix,iy,iw,ih:  iexp
ox,oy ow oh:  ivar
```

Esta función calcula la extensión total de una ventana del tamaño a partir de la zona de trabajo o inversamente el tamaño de la zona de trabajo a partir del tamaño de las ventana.

```
rueck     0 = Surgió un error.
w_type    0 = Calcular la extensión total.
          1 = Calcular la zona de trabajo.
```

attr	Partes de la ventana:			Bit-Nro
	&H0001	NAME	Título con el nombre.	0
	&H0002	CLOSE	Campo de cierre.	1
	&H0004	FULL	Campo del tamaño máximo.	2
	&H0008	MOVE	Barra de desplazamiento.	3
	&H0010	INFO	Línea Info.	4
	&H0020	SIZE	Ajuste del tamaño de la ventana.	5
	&H0040	UPARROW	Flecha hacia arriba.	6

&H0080	DNARROW	Flecha hacia abajo.	7
&H0100	VSLIDE	Posicionador vertical.	8
&H0200	LFARROW	Flecha hacia la izquierda.	9
&H0400	RTARROW	Flecha hacia la derecha.	10
&H0800	HSLIDE	Posicionador horizontal.	11

ix
iy
iw
ih Coordenadas conocidas.

ox
oy
oww
oh Coordenadas calculadas.

Biblioteca de resource (resource library)

Contiene rutinas para crear una superficie gráfica del usuario, que independientemente de la resolución elegida simplifica el intercambio de datos entre el usuario y el programa.

RSRC_LOAD(name$)

name$: sexp

Esta función reserva espacio de memoria y carga un archivo de resource. A continuación se fijan señaladores internos y se convierten coordenadas de formato de caracteres en formato de pixel. Para resources definidas por uno mismo en la memoria ha de aplicarse RSRC_OBFIX.

rueck 0 = Surgió un error.
name$ Camino de acceso del correspondiente archivo resource.

Ejemplo:

```
~RSRC_LOAD("TEST.RSC")
```

RSRC_FREE()

Esta función vuelve a liberar el espacio de la memoria reservado con RSRC_LOAD.

rueck 0 = Surgió un error.

Ejemplo:

```
~RSRC_FREE()
```

RSRC_GADDR(type,index,adr)

type,index: iexp
adr: ivar

Ejemplo:

```
~SHEL_WRITE(1,1,1,"","GFABASIC.PRG")
```

--> Esto significa, que después de abandonar el BASIC y de retornar al desktop se vuelve a iniciar el GFA-BASIC.

```
SHEL_GET(anz,x$)
```

```
anz:  iexp
x$: svar
```

Esta función hace posible la lectura de la memoria (desktop.inf).

```
rueck    0 = Surgió un error.
anz      Cantidad de bytes a leer.
x$       String para captar los caracteres leídos.
```

```
SHEL_PUT(len,x$)
```

```
len: iexp
x$:  sexp
```

Esta función hace posible escribir en la memoria environment del GEM (desktop.inf) una cantidad de caracteres a indicar.

```
rueck    0 = Surgió un error.
x$       String, que contiene los caracteres a escribir.
len      Cantidad de bytes a escribir.
```

Ejemplo:

```
' anunciar GFA-BASIC
~SHEL_GET(2000,a$)
q%=INSTR(a$,CHR$(26))
IF q%
  a$=LEFT$(a$,q%-1)
  IF INSTR(a4,"GFABASIC.PRG")=0
    a$=a$+"#G 03 04   c:\GFA\GFABASIC.PRG@ *.GFA@ "+MKI$(&hd0a)+CHR$(26)
    ~SHEL_PUT(LEN(a$),a$)
  ENDIF
ENDIF
```

--> El programa avisa que C:\GFA\GFABASIC.PRG está conectado, o sea que con cada activación doble (click) de files con el sufijo .GFA se inicia este GFA-BASIC.

Atención: Asegurar trabajo por medio de:

```
~SHEL_GET(3000,a$)
OPEN"O",#1,"C:\DESKTOP.INF"
PRINT #1,LEFT$(A$,INSTR(A$,CHR$(26))
CLOSE #1
' Importante esel caracter CHR$(26)
```

```
SHEL_FIND(camino$)

camino$: svar
```

Esta función busca un archivo en la carpeta actual, en el índice básico de la disquetera activa y de la disquetera A: y en caso exitoso evidencia el camino completo a este archivo (o bien xx.x o bien \xx.x o a:\xx.x)

rueck	0 = Surgió un error.
camino$	String que contiene el nombre del archivo buscado. Finalizada la función este string contiene el camino de acceso completo al archivo.

```
SHEL_ENVRN(adr,such$)

adr:  avar
such$:  sexp
```

Esta función sirve para determinar valores de variables en el Environment GEM.

rueck	Reservado, siempre 1.
adr	Dirección, delante de la cual figura el string indicado en such$.
such$	String que contiene los parámetros de búsqueda (por ej. A:\).

Ejemplo:

```
PRINT SHEL_ENVRN(a%,"P")
PRINT CHAR{a%-1}
```

--> Se evidencia: PATH=;A:\

Programas de ejemplo

En este apartado se muestran varios programas de ejemplo y se explica brevemente sus funciones.

```
'
' Biblioteca gráfica
'
REPEAT
  CLS
  PRINT "rubber = 1, drag = 2, move = 3, grow_shrink = 4, ende = 0"
  INPUT "wahl: ";wahl%
  ON wahl% GOSUB rubber,drag,move,grow_shrink
UNTIL wahl%=0
'
EDIT
'
PROCEDURE rubber
  GRAPHMODE 3
  DEFFILL 1,2,4
```

```
    REPEAT
      MOUSE mx%,my%,mk%
      IF mk% AND 1
        x1%=mx%
        y1%=my%
        ~GRAF_RUBBERBOX(x1%,y1%,16,16,lx%,ly%)
        PBOX x1%,y1%,x1%+lx%,y1%+ly%
      ENDIF
    UNTIL mk% AND 2
  RETURN
  '
  PROCEDURE drag
     GRAPHMODE 1
    DEFFILL 1,2,4
    REPEAT
      MOUSE mx%,my%,mk%
      BOX 50,50,200,150
      BOX 40,40,400,300
      IF mk% AND 1
        ~GRAF_DRAGBOX(150,100,50,50,40,40,400,300,lx%,ly%)
        PBOX lx%,ly%,lx%+150,ly%+100
      ENDIF
    UNTIL mk% AND 2
  RETURN
  '
  PROCEDURE move
    GRAPHMODE 1
    DEFFILL 1,2,4
    b%=64
    h%=64
    FOR i%=0 TO 639-b% STEP b%
      FOR j%=0 TO 399-h% STEP h%
        ~GRAF_MOVEBOX(b%,h%,i%,j%,639-i%,399-j%)
      NEXT j%
    NEXT i%
  RETURN
  '
  PROCEDURE grow_shrink
    GRAPHMODE 1
    ~GRAF_GROWBOX(319,199,16,16,0,0,639,399)
    PRINT "Esto fue graf_growbox"
    PRINT "Oprima una tecla"
    ~INP(2)
    ~GRAF_SHRINKBOX(319,199,16,16,0,0,639,399)
    PRINT "Esto fue graf_shrinkbox"
    PRINT "Oprima una tecla"
    ~INP(2)
  RETURN
```

--> Pregunta por la rutina deseada y bifurca a la respectiva procedure.
    1 : El programa dibuja un rectángulo a partir del momento en que se oprime el botón izquierdo del ratón. Manteniendolo apretado usted puede variar el tamaño del rectángulo. Al dejar de pulsar el botón el rectángulo es rellenado con un color. Puede salir de esta rutina apretando el botón derecho del ratón.
    2 : El programa dibuja un gran rectángulo y dentro de él un segundo rectángulo. Mientras que mantenga pulsado el botón izquierdo del ratón puede mover el segundo rectángulo dentro del contorno del primer rectángulo. Luego el rectángulo es rellenado. Puede salir de la rutina

pulsando el botón derecho del ratón.
3 : El programa bifurca a la procedure GRAF_MOVEBOX, que hace girar una serie de rectángulos alrededor del centro de la pantalla.
4 : Es dibujado un rectángulo que se extiende y luego de apretar una tecla vuelve a reducirse. Puede interrumpir el programa apretando 0.

```
    ' Administración de boxes de diálogo
'
DIM r%(3)
'           Indices de resource para DIALOG
form1%=0    !Formular/Dialog
ikon1%=1    !ICON en árbol FORM1
vor%=2      !FTEXT en árbol FORM1
nac%=3      !FTEXT en árbol FORM1
str%=4      !FTEXT en árbol FORM1
ort%=5      !FTEXT en árbol FORM1
abbruch%=6  !BUTTON en árbol FORM1
ok%=7       !BUTTON en árbol FORM1
r%(1)=8     !BUTTON en árbol FORM1
r%(2)=9     !BUTTON en árbol FORM1
r%(3)=10    !BUTTON en árbol FORM1
ausgabe%=11 !STRING en árbol FORM1
'
    ~RSRC_FREE()
RESERVE FRE(0)-32000
laden_ok!=RSRC_LOAD("\dialog.rsc") ! cargar Resource
IF NOT laden_ok!
  ALERT 1,"no encuentro el RSC",1," Return ",a%
  RESERVE FRE(0)+32000
  EDIT
ENDIF
~RSRC_GADDR(0,0,tree%)    !Determinar la dirección del árbol de objetos
~FORM_CENTER(tree%,x%,y%,w%,h%)    !Centrar coordinadas del objeto
'
' Ocupar los campos para editar con textos
CHAR{{OB_SPEC(tree%,vor%)}}="Johann Sebastian"
CHAR{{OB_SPEC(tree%,nac%)}}="Bach"
CHAR{{OB_SPEC(tree%,str%)}}="Kantatengasse 77"
CHAR{{OB_SPEC(tree%,ort%)}}="Dresden"
'
~OBJC_DRAW(tree%,0,1,x%,y%,w%,h%)  !Dibujar árbol de objetos
'
REPEAT
  ex%=FORM_DO(tree%,0)  !Fue seleccionado un objeto con estado Exit-Status ?
  '
  ' Leer textos de los campos
  vorname$=CHAR{{OB_SPEC(tree%,vor%)}}
  nachname$=CHAR{{OB_SPEC(tree%,nac%)}}
  strasse$=CHAR{{OB_SPEC(tree%,str%)}}
  ort$=CHAR{{OB_SPEC(tree%,ort%)}}
  '
  FOR i%=1 TO 3
    IF BTST(OB_STATE(tree%,r%(i%)),0)   !cual Radio-Button
      radio%=r%(i%)                     !fue seleccionado ?
    ENDIF
  NEXT i%
```

```
UNTIL ex%=ok% OR ex%=abbruch%
'
~RSRC_FREE()   !reservierten Speicherplatz wieder freigeben
RESERVE FRE(0)+32000
'
CLS
PRINT "Salió con : ";ex%
PRINT "Nombre    : ";vorname$
PRINT "Apellido  : ";nachname$
PRINT "Dirección : ";strasse$
PRINT "Ciudad    : ";ort$
PRINT "Radio     : ";radio%
```

--> El archivo DIALOG.RSC es cargado. y determinada la dirección. Luego las coordenadas del objeto son centradas y los campos para editar son ocupados con strings. Esto sucede por medio de OB_SPEC.

Esta función devuelve un señalador a una estructura TEDINFO, en la cual están almacenadas informaciones sobre un texto, como p. ej. dirección del texto, el argo, etc. Esta estructura contiene por su parte otro señalador que indica al string en sí.
Objeto --> Estructura Tedinfo --> String

El árbol de objetos es dibujado, pasando luego a la estructura de repetición REPEAT UNTIL, que controla periodicamente los objetos por medio de FORM_DO. El programa comprueba si fue seleccionado un objeto con estado EXIT y en ese caso abandona la estructura de repetición. Al final indica el número del botón con que salió de la repetición, los datos que usted entró y el número del botón de radio seleccionado.

```
' Programación de la barra de menú
'
' Reservar memoria para el Resource, cargar Resource
' y visualizar menú.
RESERVE FRE(0)-33000
IF RSRC_LOAD("\handbuch.rsc")=0
  ALERT 1,"No encuentro|el archivo de Resource.",1," Return ",a%
  RESERVE FRE(0)+33000
  EDIT
ENDIF
~RSRC_GADDR(0,0,menu_adr%)
~MENU_BAR(menu_adr%,1)
'
' Instalar Buffer para mensajes y preparar variables
DIM message_buffer%(3)
mes_adr%=V:message_buffer%(0)
ABSOLUTE mes_type&,mes_adr%
ABSOLUTE m_titel&,mes_adr%+6
ABSOLUTE m_eintrag&,mes_adr%+8
'
REPEAT
  ~EVNT_MULTI(&X110000,0,0,0,0,0,0,0,0,0,0,0,0,0,0,mes_adr%,100)
  ' si fue seleccionado un item de menú
  IF mes_type&=10
    ' si fue seleccionado un item diferente al anterior
    IF obj_nr%<>m_eintrag&
```

```
            obj_nr%=m_eintrag&
            titel$=CHAR{OB_SPEC(menu_adr%,m_titel&)}
            eintrag$=CHAR{OB_SPEC(menu_adr%,m_eintrag&)}
            PRINT AT(3,20);"Titulo de menú:     ";titel$;SPC(15)
            PRINT AT(3,22);"Item de menú  :     ";eintrag$;SPC(15)
            ~MENU_TNORMAL(menu_adr%,m_titel&,1)
          ENDIF
        ENDIF
      UNTIL MOUSEK=2
      '
      ' Borrar barra de menú, eliminar el Resource de la memoria
      ' y liberar la memoria reservada
      ~MENU_BAR(menu_adr%,0)
      ~RSRC_FREE()
      RESERVE FRE(0)+33000
      END
```

--> Al comienzo del programa son cargados 33 Kilobyte de Resource después de haber reservado memoria. Esta memoria reservada es luego liberada al terminar el programa.
Luego de cargar el archivo es determinada la dirección del árbol de menú y visualizado.
La administración del menú sucede por medio de EVNT_MULTI. Si selecciona un item de menú, ésto es registrado en el buffer de mensajes. Este está compuesto por 16 Bytes en forma de 8 words.
En los primeros dos Bytes es registrado el número 10 si fue seleccionado un item. A partir del sexto byte se encuentra el número de objeto del item.

Usted puede salir de la estructura de repetición apretando el botón derecho del ratón. La función EVNT_MULTI controla el menú. En las variables m_titel& y m_eintrag& se encuentran los números de objeto del título de menú y item seleccionados. OB_SPEC señala a una estructura TED_INFO, en la cual en los primeros 4 bytes se encuentra otro señalador que muestra sobre el texto del objeto. De esta manera puede determinar el texto del item seleccionado.

Después de escribir ambos textos en la pantalla, el título del menú es reinvertido pasando al estado normal.

```
' Administración de ventanas
'
DEFFILL 1,2,4
PBOX 0,19,639,399
DEFFILL 1,0
'
DIM message_buffer%(3)  ! 16 Byte
adr_mes%=V:message_buffer%(0)
'
ABSOLUTE word0&,adr_mes%
ABSOLUTE x&,adr_mes%+8
ABSOLUTE y&,adr_mes%+10
ABSOLUTE w&,adr_mes%+12
ABSOLUTE h&,adr_mes%+14
'
handle&=WIND_CREATE(&X101111,0,19,639,380)
'
```

```
titel$="Window"
adr_tit%=V:titel$
~WIND_SET(handle&,2,CARD(SWAP(adr_tit%)),CARD(adr_tit%),0,0)
~WIND_OPEN(handle&,100,100,200,100)
~WIND_GET(handle&,4,wx&,wy&,ww&,wh&)
PBOX wx&,wy&,wx&+ww&,wy&+wh&
'
raus!=FALSE
REPEAT
  ' la próxima linea solo fue dividida por razones de espacio
  ~EVNT_MULTI(&X110000,0,0,0,0,0,0,0,0,0,0,0,0,0,adr_mes%,100)
  SELECT word0&
  CASE 22                                   ! WM_CLOSED
    raus!=TRUE
  CASE 23                                   ! WM_FULLED
    ~WIND_SET(handle&,5,1,19,638,380)
    ~WIND_GET(handle&,4,wx&,wy&,ww&,wh&)
    PBOX wx&,wy&,wx&+ww&,wy&+wh&
    word0&=0
  CASE 27,28                                ! WM_SIZED, WM_MOVED
    IF w&<100
      w&=100
    ENDIF
    IF h&<80
      h&=80
    ENDIF
    ~WIND_SET(handle&,5,x&,y&,w&,h&)
    ~WIND_GET(handle&,4,wx&,wy&,ww&,wh&)
    PBOX wx&,wy&,wx&+ww&,wy&+wh&
    word0&=0
  ENDSELECT
UNTIL raus!
~WIND_CLOSE(handle&)
~WIND_DELETE(handle&)
```

--> Al principio del programa es instalado un Buffer para mensajes, en el cual deben ser escritas todas las selecciones hechas a elementos de alteración de ventana.
La ventana es creada con WIND_CREATE. WIND_SET le fija un nombre. Después de abrir la ventana con WIND_OPEN el programa lee los datos de la superficie de trabajo y cubre la superficie con un rectángulo en blanco.
En la siguiente estructura de repetición los elementos de alteración de ventana son controlados por la función EVNT_MULTI. Los diferentes casos son tratados en CASE. El programa finaliza si selecciona el elemento de cierre de ventana.

12 - Anexo

Compatibilidad con el GFA-BASIC 2.xx

En el GFA-BASIC 3.0 es posible emplear programas de versiones anteriores de GFA-BASIC. Para eso, se deben almacenar estos programas con la versión antigua como archivo ASCII (Save,A) y se deben cargar en la versión nueva con Merge. Una vez hecha esta transferencia se puede seguir usando el programa en GFA-BASIC 3.0 con Save y Load.

Todas las instrucciones contenidas en versiones anteriores del intérprete Basic también se encuentran en la versión 3.0. Pero en parte existen pequeñas desviaciones en el significado de la instrucción, que requieren de una adaptación.

MUL DIV

En la versión 3.0 las instrucciones MUL y DIV elaboran con variables enteras (I,&,%) solo parámetros enteros.

En las versiones anteriores

```
a%=10
MUL a%,2.5
```

llevaba a que en a% figurara el valor 25. Sin embargo, la versión 3.0 no respeta los lugares detrás de la coma en los parámetros de la instrucción MUL, de tal manera que a% es igual a 20. A cambio de esta incompatibilidad, las instrucciones mencionadas son mucho más veloces que en versiones anteriores de GFA-BASIC debido al empleo de una verdadera aritmética de enteros. Aunque esto vale únicamente para variables enteras.

PRINT USING

En caso de desbordamiento, PRINT USING no reporta el número no formateado, sino únicamente los números que caben en el formato correspondiente y con eso eventualmente valores falsos en lugares de formatos falsos. Además, en el formato exponencial también se tratan muchos símbolos "^" en forma correcta y cuando se trata de varios lugares antes de la coma se realiza un ajuste del exponente.

CLS-PRINT TAB

Con CLS se evidencia ahora ESC-E-CR, de tal manera que el primer PRINT TAB es tratado correctamente.

KEYPAD

Un programa, que solicita las teclas del bloque numérico con Alternate y/o Control, necesita ahora un KEYPAD 0 para desconectar la instalación especial de teclas. Lo mismo vale para las teclas de función con Alternate.

MOUSEX-MOUSEY

Cuando hay ventanas activas y se solicitan MOUSEX/MOUSEY, resultan coordenadas negativas por encima y a la izquierda de los límites de la

ventana (CLIP OFFSET resp.). En la versión 2.X se retornaba CARD(MOUSEX) o bien CARD(MOUSEY).

Luego se pueden acelerar notoriamente los programas generados en el intérprete si se emplea el compilador del GFA-BASIC. De esta forma surgen programas que también se pueden correr sin el intérprete. Además es posible la aplicación del intérprete RUN-ONLY contenido en el disco suministrado. Este puede ser agregado a programas propios y difundido por aquellos compradores del GFA-BASIC que estén inscriptos. Así es posible el uso de programas en GFA-BASIC para usuarios sin intérprete de GFA-BASIC propio.

Tabla GEMDOS

~GEMDOS(0) pterm0()

Finaliza un programa. No se debe usar en GFA-BASIC.

r%=GEMDOS(1) cconin()

Leer carácter del teclado con indicación en la pantalla (comparar INP(2)).

r% Bit 0-7: Código ASCII, 16-23: Código Scan, 24-31: Teclas de conmutación.

~GEMDOS(2,z%) cconout()

Evidenciar carácter en la pantalla (comparar OUT2,z%).

z% Bit 0-7: Código ASCII del carácter.

r%=GEMDOS(3) cauxin()

Lee un carácter de la interfase seriada (comparar INP(1)).

r% Código del carácter leído.

~GEMDOS(4,z%) cauxout()

Evidenciar carácter en la interfase seriada (comparar OUT1,z%).

z% Código del carácter a evidenciar.

~GEMDOS(5,z%) cprnout()

Evidenciar caracteres en la interfase de la impresora (comparar OUT0,z%).

z% Código del carácter a evidenciar.
r%=GEMDOS(6,z%) crawio()

Leer o escribir carácter del teclado sin indicación en la pantalla (comparar OUT 2,z% así como INKEY$).

r% con z%=255 el carácter leído por el teclado.
z% Código del carácter a evidenciar, con z%=255 leer carácter.

r%=GEMDOS(7) crawcin()

Leer carácter del teclado sin indicación en la pantalla (comparar INP(2)).

r% Código del carácter leído.

r%=GEMDOS(8) cnecin()

Igual que GEMDOS(7), pero sin tener en cuenta caracteres de control, por ej. CTRL + C.

r% Código del carácter leído.

~GEMDOS(9,L:adr%) cconws()

Evidencia una cadena de caracteres en la pantalla.

adr% Dirección del string, que debe terminar con el byte cero.

~GEMDOS(10,L:adr%) cconrs()

Lee e incorpora una cadena de caracteres del teclado (CTRL+C conduce a la interrupción).

adr% Buffer para string: el primer byte es la cantidad de caracteres a leer, el segundo byte es la cantidad de caracteres leídos, luego sigue el string.

r!=GEMDOS(11) cconis()

Revisa, si hay un carácter en el buffer del teclado.

r! Es TRUE, cuando existe un carácter, sino es FALSE.

~GEMDOS(14,d%) dsetdrv()

Fija la disquetera actual; 0=A,1=B, etc. (comparar CHDRIVE).

d% Número de disquetera.

r!=GEMDOS(16) cconos()

Revisa, si en la pantalla se puede evidenciar un carácter.

r! Estado de salida, siempre es TRUE.

r!=GEMDOS(17) cprnos()

Revisa, si se puede evidenciar en la interfase paralela.

r! Estado de salida, es TRUE cuando puede recibir (normalmente impresora).

r!=GEMDOS(18) cauxis()

Revisa, si existe un carácter en la interfase seriada.

r! TRUE cuando existe un carácter, sino es FALSE.

r!=GEMDOS(19) cauxos()

Revisa el estado de salida de la interfase seriada.

r! TRUE, si se puede evidenciar un carácter, sino es FALSE.

r%=GEMDOS(25) dgetdrv()

Determina la disquetera actual.

r% Número de la disquetera actual (A=0, B=1, etc.)

~GEMDOS(26,L:adr%) fsetdta()

Fija la dirección de transferencia de disco (DTA), normalmente BASEPAGE+128.

adr% La dirección que se debe fijar.

r%=GEMDOS(42) tgetdate()

Calcula la fecha del sistema (comparar DATE$).

r% Fecha: Bit 0-4: día, 5-8: Mes, 9-15: Año menos 1980.

~GEMDOS(43,d%) tsetdate()

Fija la fecha (comparar SETTIME).

d% La fecha nueva (formato ver GEMDOS (42)).

r%=GEMDOS(44) tgettime()

Calcula la hora del sistema (comparar TIME$).

r% Hora: Bit 0-4: segundos, 5-10: minutos, 11-15: segundos.

~GEMDOS(45,t%) tsettime()

Fija la hora del sistema (comparar SETTIME).

t% La nueva hora (formato ver GEMDOS(44)).

r%=GEMDOS(47) fgetdta()

Calcula la dirección actual de transferencia de disco (DTA).

r% La dirección calculada.

r%=GEMDOS(48) sversion()

Calcula el número de versión GEMDOS.

r% El número de versión.

!GEMDOS(49,L:b%,r%) pterm0res()

Finaliza el programa y lo mantiene residiendo (en GFA-BASIC no se puede usar).

b% Cantidad de bytes a mantener residiendo a partir de BASEPAGE.
r% Valor de transferencia al programa a llamar.

r%GEMDOS(54,L:adr%,d%) dfree()

Calcula datos a través del espacio de memoria de un disco (comparar DFREE).

r% -46 si se indicó un número de disquetera erróneo.
adr% Dirección de una estructura Disk-Info de 4 por 4 bytes de tamaño:
Long 1: Cantidad de cluster libres, Long 2: Cantidad total de cluster, Long 3: Bytes por sector, Long 4: sectores por cluster.

r%=GEMDOS(57,L:adr%) dcreate()

Ubica una carpeta en el índice actual (comparar MKDIR).

r% -34 o -36 si surgió un error.
adr% Dirección del nuevo nombre de lacarpeta (debe terminar en byte cero).

r%=GEMDOS(58,L:adr%) ddelete()

Borra un ordenador (comparar RMDIR).

r% -34,-36 o -65 si surgió un error.
adr% Dirección del nombre del ordenador (debe terminar en byte cero).

r%=GEMDOS(59,L:adr%) dsetpath()

Cambia el índice actual.

r% -34 si no se halló el nuevo índice.
adr% Dirección del nombre del índice (debe terminar en byte cero).

r%=GEMDOS(60,L:adr%,a%) fcreate()

Ubica un nuevo archivo (comparar OPEN"O").

r% -34,-35 o -36 si surgió un error.
adr% Dirección del nombre del archivo (debe terminar en byte cero).
a% Se fijó bit 0: archivo protegido, no se puede escribir, Bit 1: Archivo escondido, Bit 2: Archivo de sistema (también escondido),
Bit 3: Nombre del disco.

r%=GEMDOS(61,L:adr%,m%) fopen()

Abre un archivo (comparar con OPEN).

r% -33,-35 o -36 en caso de error, sino handle del archivo.
adr% Dirección del nombre del archivo (debe terminar en byte cero).
m% 0 para leer, 1 para escribir, 2 para leer y escribir.

r%=GEMDOS(62,h%) fclose()

Cierra un archivo (comparar CLOSE).

```
r%    -37 si surgió un error.
h%    Handle del archivo a cerrar.
```

```
r%=GEMDOS(63,h%,L:I%,L:adr%)                    fread()
```

Lee bytes de un archivo abierto (comparar BGET).

```
r%      -37 en caso de error, sino cantidad de bytes leídos.
h%      Handle del archivo.
l%      cantidad de bytes a leer.
adr%    Dirección a la cual se deben escribir los bytes.
```

```
r%=GEMDOS(64,h%,L:I%,L:adr%)                    fwrite()
```

Escribe bytes en un archivo (comparar BPUT).

```
r%     -36 o -37 en caso de error, sino cantidad de bytes escritos.
h%     Handle del archivo.
l%     Cantidad de bytes a escribir.
adr%   Dirección, a partir de la cual figuran en la memoria los bytes
       a escribir.
```

```
r%=GEMDOS(65,L:adr%)                            fdelete()
```

Borra un archivo (comparar KILL).

```
r%    -33 o -36 si surgió un error.
adr%  Dirección del nombre del archivo (debe finalizar con byte cero).
```

```
r%=GEMDOS(66,L:n%,h%,m%)                        fseek()
```

Vuelve a fijar de nuevo el señalador para accesos a archivos (comparar SEEK, RELSEEK).

```
r%   -32 o -37 si surgió un error.
n%   Cantidad de bytes a saltear.
m%   0: a partir del comienzo del archivo, 1: a partir de la posición
     actual, 2: a partir del final del archivo.
```

```
r%=GEMDOS(67,L:adr%,m%,a%)                      fattrib()
```

Lee o escribe atributos de un archivo.

```
r%     -33 o -34 en caso de error, sino atributos del archivo que se
       usaron hasta el momento.
adr%   Dirección del nombre del archivo (debe finalizar con byte cero).
m%     0: Leer atributos del archivo, 1: escribir atributos del ar-
       chivo.
a%     Atributos del archivo:  Bit 0: protegido, no se puede escribir,
       1: escondido, 2: Archivo de sistema, 4: ordenador, 5: bit de ar-
       chivo.
```

```
r%=GEMDOS(69,h%)                                fdup()
```

Genera un segundo handle del archivo.

```
r%   -35 o -37 en caso de error, sino segundo handle del archivo.
h%   El handle original del archivo.
```

```
r%=GEMDOS(70,h%,nh%)                                    fforce()
```

Fija un nuevo handle de salida para datos que salen del GMDS.

```
r%    -37 si surgió un error.
h%    Handle del canal de datos a desviar.
nh%   Handle del canal, al cual se desvían los datos.
```

```
r%=GEMDOS(71,L:adr%,d%)                                 dgetpath()
```

Calcula el camino de acceso actual de una disquetera (comparar DIR$).

```
r%      -46 si surgió un error.
adr%    Dirección a partir de la cual se debe depositar el camino de
        acceso.
d%      Caracterización  de la disquetera (0=actual, 1=A, 2=B,etc.).
```

```
r%=GEMDOS(72,L:b%)                                      malloc()
```

```
r%    para b%=-1 longitud del mayor espacio de memoria libre, sino
      dirección inicial de la zona reservada o mensaje de error.
b%    Cantidad de bytes a reservar, para b%=-1 ver r%.
```

```
r%=GEMDOS(73,L:adr%)                                    mfree()
```

Libera una zona reservada con GEMDOS(72) (comparar MFREE).

```
r%    -40 en caso de error.
adr%  Dirección de la zona de memoria a liberar.
```

```
r%=GEMDOS(74,L:adr%,L:b%)                               mshrink()
```

Acorta una zona reservada con GEMDOS(72) (comparar MSHRINK).

```
r%    -40 o -67 si surgió un error.
adr%  Dirección de la zona de memoria a achicar.
b%    Nueva longitud de la zona de memoria en bytes.
```

```
r%=GEMDOS(75,m%,L:p%,L:c%,L:e%)                         pexec()
```

Ejecuta un programa como subprograma (comparar EXEC).

```
r%    -32, -33, -39 o -66 en caso de error.
m%    0= Cargar e iniciar, 3: Cargar, 4:Iniciar, 5: generar Basepage.
p%    Dirección del nombre del programa o en m%=4 el basepage.
c%    Dirección de la línea de comando ( no en m%=4)
e%    Dirección del string de environment ( no en m%=4)
```

```
~GEMDOS(76,r%)                                          pterm()
```

Finaiza el programa en ejecución (comparar QUIT, SYSTEM).

```
r%    Valor de retorno al programa que se llama.
```

```
r%=GEMDOS(78,L:adr%,a%)                                 fsfirst()
```

Busca en el índice actual archivos conb determinados nombres. El nombre hallado se deposita en DTA.

```
r%    -33: no se halló el archivo,  -49: no hay otros archivos.
adr%  Dirección del nombre del archivo (puede contener los wildcards
      * y ?).
a%    Atributos del archivo:  Bit 0: protegido, no se puede escribir,
      1: escondido, 2: archivo de sistema, 3: nombre del disco,
      4: ordenador, 5: bit del archivo.

r%=GEMDOS(79)                                          fsnext()

Continua la búsqueda iniciada con GEMDOS(78).

r%   -49 si no caben más archivos en el string de búsqueda.

r%=GEMDOS(86,0,L:o%,L:n%)                              frename()

Renombra un archivo (comparar NAME, RENAME).

r%    -34 o -36 si surgió un error.
o%    Dirección del nombre anterior del archivo .
n%    Dirección del nombre nuevo del archivo.

~GEMDOS(87,L:adr%,h%,m%)                               fdatime()

Fija o calcula la hora y fecha de un archivo (comparar TOUCH).

adr%  Dirección de la información del tiempo (4 byte).
h%    Handle del archivo.
m%    0: Leer el tiempo del archivo, 1: Fijar el tiempo del archivo.

Tabla BIOS

~BIOS(0,L:adr%)                                        getmpb()

Inicialización del memory-parameter-block.

adr%  Dirección del nuevo MPB.

r%=BIOS(1,d%)                                          constat()

Determina el estado de ingreso de una unidad (comparar INP?).

r%  0: no hay carácter disponible,  -1: carácter disponible.
d%  0: interfase paralela,  1: interfase seriada, 2: Teclado,
    3: interfase MIDI.

r%=BIOS(2,d%)                                          bconin()

Lee e incorpora un carácter de una unidad (comparar INP).

r%   Carácter leído e incorporado (8 bit).
d%   0: interfase paralela,  1: interfase seriada, 2: Teclado,
     3: interfase MIDI.

~BIOS(3,d%,b%)                                         bconout()

Evidencia un carácter sobre unidad (comparar OUT).

d%   0: interfase paralela,  1: interfase seriada, 2: Teclado,
     3: interfase MIDI, 4: IKBD.
```

```
b%   El carácter a evidenciar.

r%=BIOS(4,f%,L:b%,n%,s%,d%)                    rwabs()

Leer y escribir sectores sobre disco.

r%     0 si surgió un error.
f%     0: leer, 1: escribir.
b%     Dirección de la zona de memoria con los datos.
n%     Cantidad de sectores.
s%     Número del sector de iniciación.
d%     Número de la disquetera (0=A, 1=B,etc.)

r%=BIOS(5,n%,L:adr%)                           setexec()

Fijar y leer vectores de escepción.

r%     Para adr%=-1 el valor del vector hasta el momento.
n%     Número del vector de excepción.
adr%   Nueva dirección del vector o -1 (ver r%).

r%=BIOS(6)                                     tickcal()

Solicita el timer del sistema.

r%     Cantidad de milisegundos transcurridos (resolución en 20ms).

BIOS(7,d%)                                     getbpb()

Calcula la dirección del bloque de parámetros del BIOS de una
disquetera.

r%      Dirección del BPB.
d%      Número de disquetera (0=A, 1=B,etc.).

R!=BIOS(8,d%)                                  bcostat()

Calcula el estado de salida de una unidad (comparar OUT?).

r!     TRUE si se puede enviar carácter, sino FALSE.
d%     0: Interfase paralela, 1: interfase seriada, 3: interfase MIDI,
       4: clip del teclado (IKB).

r%=BIOS(9,d%)                                  mediach()

Revisa, si ha sido cambiado el disco.

r%     0: seguro que no, 1: quizás, 2: cambiado.
d%     Número de disquetera (0=A, 1=B,etc,).

r%=BIOS(10)                                    drvmap(()

Revisa, que disqueteras están acopladas.

r%     Vector de bit, bit fijado por disquetera (Bit 0=A, Bit 1=B,
       etc.).

r%=BIOS(11,c%)                                 kbshift()
```

Calcula o fija el estado de las teclas de consutación.

r% Para c%=-1 estado actual de las teclas de conmutación.
c% Nuevo estado, Bit 0: Shift izquierda, Bit 1: Shift derecha,
Bit 2: Control, Bit 3: Alternate, Bit 4: Caps-Lock,
Bit 5: Alternate+Clr/Home (tecla derecha del ratón),
Bit 6: Alternate+Insert (tecla izquierda del ratón).

Tabla XBIOS

~XBIOS(0,t%,L:p%,L:v%) initratón()

Inicializa rutinas del ratón (no compatible con GEM).

t% 0: desconectar ratón, 1: conectar ratón en modo relativo,
2: conectar ratón en modo relativo, 4: ratón en modo de teclado.

p% Dirección de una estructura de información.
v% Dirección de una rutina de tratamiento del ratón.

r%=XBIOS(2) physbase()

Calcula la dirección de la memoria de la pantalla.

r% Dirección de la memoria física de la pantalla.

r%=XBIOS(3) logbase()

Calcula la dirección de la memoria lógica de la pantalla.

r% Dirección de la memoria lógica de la pantalla.

r%=XBIOS(4) getrez()

Calcula la resolución actual de la pantalla.

r% 0: 320x200, 1: 640*200, 2: 640x400, 3: reservado.

~XBIOS(5,L:l%,L:p%,r%) setscreen()

Fija las direcciones de las memoria de la pantalla y la resolución, el valor -1 significa en todos los parámetros no hubo modificación. Direcciones solo en límites de 256 byte.

l% Nueva dirección de la memoria lógica de la pantalla.
p% Nueva dirección de la memoria física de la pantalla.
r% Nueva resolución de la pantalla (ver XBIOS(4), no compatible con GEM).

~XBIOS(6,L:adr%) setpalette()

Carga una nueva paleta de colores.

adr% Dirección de una tabla de 16 palabras, que contiene la nueva paleta de colores.

r%=XBIOS(7,n%,c%) setcolor()

Fijar o solicitar un registro de colores (comparar SETCOLOR).

```
r%   Para c%=-1 solicita el registro de colores indicado.
n%   Número del registro de colores (0 a 15).
c%   Nuevo color, con c%=-1 ver r%.

r%=XBIOS(8,L:b%,d%,sec%,t%,side%,n%)          floprd()

Lee sectores de un diskette.

r%     0 si surgió un error.
b%     Dirección de la zona, en la cual se leen los sectores.
f%     Sin usar.
d%     Número de disquetera (0=A, 1=B,etc.).
sec%   Número de sector, a partir del cual se lee.
t%     Número de pista (track), de la cual se lee.
side%  Lado del disco (0 o 1).
n%     Cantidad de sectores a leer (deben figurar sobre un track).

r%XBIOS(9,L:b%,L:f%,d%,sec%,t%,side%,n%)        flopwr()

Escribe sectores sobre disco.

r%    0 si surgió un error.
b%    Dirección de la zona de memoria con los datos a escribir.
f%    Sin usar.
d%    Número de disquetera (0=A, 1=B,etc.).
sec%  Número de sector, a partir del cual se escribe.
t%    Número de pista (track), en el cual se escribe.
side% Lado del disco (0 o 1).
n%    Cantidad de sectores a escribir (todos sobre un track).

r%=XBIOS(10,L:b%,L:f%,d%,sec%,t%,side%,i%,L:m%,v%)   flopfmt()

Formatea una pista del disco.

r%     0 si surgió un error.
b%     Dirección de una zona para almacenamientos intermedios
       (min.8KB).
f%     Sin usar.
d%     Número de disquetera (0=A, 1=B,etc.).
sec%   Sectores por track (normalmente 9).
t%     Número de pista a formatear (track).
side%  Lado del disco (0 o 1).
i%     Factor Interleave (normalmente 1).
m%     Magic number &H87654321.
v%     Valor en sectores después del formateo (normalmente &HE5E5).

~XBIOS(12,n%,L:adr%)                          midiws()

Evidencia una zona de memoria de la interfase MIDI.

n%     Cantidad de bytes a evidenciar menos 1.
adr%   Dirección de la zona de memoria a evidenciar.

~XBIOS(13,n%,L:adr%)                          mfpint()

Modificar vectores Interrupt del MFP ( no se puede usar en GFA-BASIC).

n%     Número de interrupts (0 a 15).
adr%   Nueva dirección del interrupt a conectar.
```

```
r%=XBIOS(14,d%)                                    iorec()

r%    Dirección del buffer de datos para el ingreso y salida de la
      unidad.
d%    0: RS 232, 1: IKBD,  2: MIDI.

~XBIOS(15,b%,h%,ucr%,rsr%,tsr%,scr%)               rsconf()

Fija los parámetros para la interfase seriada.

b%    Baudrate.
h%    Handshake-modus,  0=sin,  1: XON/XOFF,  2: RTS/CTS,  3: ambos.
ucr%  Registro de control USART  del MFP.
rsr%  Receiver-Registro de estado del MFP.
tsr%  Transmitter-Registro de estado del MFP.
scr%  Synchronus-Registro de caracteres del MFP

r%=XBIOS(16,L:us%,L:sh%,L:cl%)                     keytbl()

Modificación de las direcciones de las tablas de instalación del
teclado.

r%     Dirección de la estructura KEYTAB.
us%    Dirección de la tabla para teclas sin Shift.
sh%    Dirección de la tabla para teclas con Shift
cl%    Dirección de la tabla con Caps-Lock.

r%=XBIOS(17)                                       random()

Calcula un número al azar (comparar RAND, RANDOM).

r%    Un número al azar  de 24 bit ( 0 a 16777215).

~XBIOS(18,L:b%,L:s%,d%,f%)                         protobt()

Genera un sector boot sobre un disco.

b%    Dirección de un buffer (512 byte) para generar el sector boot.
s%    Número de serie , -1: tomo la vieja, fijar bit mayor que 24: al
      azar.
d%    Tipo de disco(tracks/caras). 0: 40/1, 1: 40/2, 2: 80/2,
      3: 80/2).
f%    0: Sector boot que no se puede realizar, 1: realizable,  -1: no
      modificar.

r%=XBIOS(19,L:b%,L:f%,d%,sec%,t%,side%,n%)         flopver()

Compara sectores de disco con contenido de la zona de memoria.

b%     Dirección de la zona de memoria, con la cual se debe comparar.
f%     Sin usar.
d%     Número de disquetera (0=A, 1=B,etc.).
sec%   Número de sector a partir del cual se ha de comparar.
t%     Número de pista (track).
side%  Lado de la disquetera (0 o 1).
n%     Cantidad de sectores a comparar.
```

```
~XBIOS(20)                              scrdmp()
```

Llama la rutina Hardcopy (comparar HARDCOPY).

```
r%=XBIOS(21,c%,s%)                      curscon()
```

Fija los atributos del cursor.

r% En caso de c%=5 ésto da la frecuencia de centelleo del cursor.
c% 0: desconectar el cursor, 1: conectar el cursor, 2: conectar el centelleo del cursor, 3: desconectar el centelleo, 4: ajustar la razón de centelleo al valor en s%, 5: ver r%.
s% En caso en que c%=4, fijar la razón de centelleo en s%.

```
~XBIOS(22,L:t%)                         bsettime()
```

Ajusta la fecha y la hora (comparar SETTIME).

t% Bit 0-4: segundos, 5-10: minutos, 11-15: horas, 16-20: día, 21-14: mes, 25-31: año menos 1980.

```
r%=XBIOS(23)                            bgettime()
```

Calcula la fecha y la hora (comparar TIME$,DATE$).

r% Instalación de bits ver XBIOS(22).

```
~XBIOS(24)                              bioskey()
```

Ajusta la instalación original del teclado (comparar XBIOS(16)).

```
XBIOS(25,n%,L:adr%)                     ikbdws()
```

Envía datos al chip del teclado (IKBD).

n% Cantidad de bytes a enviar menos 1.
adr% Dirección en la cual figuran los datos a enviar.

```
~XBIOS(26,i%)                           jdisint()
```

Cierra un interrupt MFP.

i% Número del interrupt a cerrar (0-15).

```
~XBIOS(27,i%)                           jenabint()
```

Libera un interrupt del MFP.

i% Número del interupt.

```
r%=XBIOS(28,d%,reg%)                    giaccess()
```

Lee y escribe registros del sound-chip.

r% Si hay acceso de lectura figura aquí el valor del registro.
d% Si hay acceso de escritura, éste es el valor a escribir (8 bit).
r% Número del registro (0 a 15), con acceso de escritura de Bit 7 fijado.

~XBIOS(29,b%) offgibit()

Borra un bit del registro PORT-A del sound-chip.

b% Diseño de bits, que se conecta con el OR existente.

~XBIOS(30,b%) ongibit()

Fija un bit del registro PORT-A del sound-chip.

b% Diseño de bits, que se conecta con el AND existente.

XBIOS(31,t%,c%,d%,L:adr%) xbtimer()

Fija los timer MFP.

t% Número del timer (0 a 3).
c% Registro de Control.
d% Registro de Data.
adr% Dirección de la rutina interrupt del timer.

~XBIOS(32,L:adr%) dosound()

Inicia una secuencia sound, que se ejecuta en el interrupt.

adr% Dirección de la zona de memoria de las instrucciones sound.

r%=XBIOS(33,c%) setprt()

Fija o lee los parámetros de la impresora.

r% Configuración actual, en caso en que c%=-1.
c% Bit no fijado/fijado, 0: Impresora de matriz/borde de tipo, 1: color/monocromático, 2: 1280/960 puntos por línea, 3: Draft/NLQ, 4: paralelo/seriado, 5: Papel infinito/hoja individual.

r%=XBIOS(34) kbdvbas()

Dirección de una tabla con vectores del procesador del teclado.

r% Dirección determinada.

r%=XBIOS(35,a%,w%) kbrate()

Fija o lee la razón de repetición del teclado.

r% Datos actuales, Bit 0-7: razón de repetición , 8-15: Tiempo de activación.
a% Retraso en la activación
w% Razón de repetición.

~XBIOS(36,L:adr%) prtblk()

Rutina hardcopy con bloque de parámetros.

adr% Dirección de un bloque de parámetros para la rutina hardcopy.

~XBIOS(37) vsync()

Espera el siguiente Vertical-blank-interrupt (comparar VSYNC).

~XBIOS(38,L:adr%) supexec()

Llamado de una rutina assembler en el modo supervisor (sin llamado del sistema).

adr% Dirección de la rutina assembler.

~XBIOS(39) puntaes()

Desconecta la AES, en la medida en que no se encuentre en el ROM.

r%=XBIOS(64,b%) blitmode()

Manejo y solicitud del blitter (solo en el blitter TOS).

r% Estado actual del blitter, cuando b%=-1, Bit 1: Está el blitter?
b% -1: ver r%, sino Bit 0: fijado el blitter está conectado, sino está desconectado, Bit 1-14: reservado (-1), Bit 15: 0.

Tabla de las variables Line-A

Con L~A se obtiene la dirección básica de las variables Line-A:

{L~A-906}	Dirección del font-header actual.
L~A-856 ..	37 palabras, DEFMOUSEN activo.
{L~A-460}	Señalador sobre el font-header actual.
L~A-456	Array de cuatro señaladores, de los cuales el último debe ser cero. Cada señalador señala sobre una lista encadenada de frases de caracteres. Los dos primeros señaladores valen para fonts residentes. El tercero es para el font-GDOS, éste se desplaza hacia atrás con cada llamado de VDI.
INT{L~A-440}	Cantidad total de estos fonts.
INT{L~A-46}	Altura de la línea de texto.
INT{L~A-44}	Máxima columna del cursor.
INT{L~A-42}	Máxima línea del cursor.
INT{L~A-40}	Longitud de una línea de texto en byte.
INT{L~A-38}	Color de fondo del texto.
INT{L~A-36}	Color del primer plano del texto.
{L~A-34}	Dirección del cursor en la memoria de la pantalla.
INT{L~A-30}	Distancia entre la primera línea de texto y el márgen superior de la pantalla.
INT{L~A-28}	CRSCOL.
INT{L~A-26}	CRSLIN.
BYTE{L~A-24}	Período de centelleo del cursor.
BYTE{L~A-23}	Contador del centelleo del cursor.
{L~A-22}	Dirección de los datos del font para el modo monocromático.
INT{L~A-18}	Ultimo carácter ASCII del font.
INT{L~A-16}	Primer carácter ASCII del font.
INT{L~A-12}	Resolución horizontal en pixel.
{L~A-10}	Dirección de la tabla de offset del font.

INT{L~A-4}	Resolución vertical en pixel.
INT{L~A-0}	Cantidad de bitplanes.
INT{L~A+2}	Bytes por línea de la pantalla.
INT{L~A+4}	Señalador sobre el campo CONTRL.
{L~A+8}	Señalador sobre el campo INTIN.
{L~A+12}	Señalador sobre el campo PTSIN.
{L~A+16}	Señalador sobre el campo INTOUT.
{L~A+20}	Señalador sobre el campno PTSOUT.
INT{L~A+24}	Valor del color para el bitplane 0.
INT{L~A+26}	Valor del color para el bitplane 1.
INT{L~A+28}	Valor del color para el bitplane 2.
INT{L~A+30}	Valor del color para el bitplane 3.
INT{L~A+32}	Flag, no dibujar el último pixel de una línea.
INT{L~A+34}	Diseño de líneas.
INT{L~A+36}	Graphmode.
INT{L~A+38}	hasta
INT{L~A+44}	2 pares de coordenadas
{L~A+46}	Señalador sobre el diseño actual de relleno.
{L~A+50}	Señalador sobre la máscara actual de relleno.
INT{L~A+52}	Flag para relleno multicolor.
INT{L~A+54}	Clipping-flag.
INT{L~A+56}	hasta
INT{L~A+64}	Coordenadas del clipping.
INT{L~A+66}	Factor de ampliación.
INT{L~A+68}	Dirección de la ampliación.
INT{L~A+70}	Flag para letra proporcional.
INT{L~A+72}	Offset x para Textblt.
INT{L~A+74}	Offset y para Textblt.
INT{L~A+76}	Coordenada x de un carácter de la pantalla.
INT{L~A+78}	Coordenada y de un carácter de la pantalla.
INT{L~A+80}	Ancho de un carácter.
INT{L~A+82}	Altura de un carácter.
{L~A+84}	Señalador sobre image de frase de caracteres.
{L~A+88}	Ancho del image de frase de caracteres.
INT{L~A+90}	Estilo del texto.
INT{L~A+92}	Máscara para la salida sombreada del texto.
INT{L~A+94}	Máscara para letra cursiva.
INT{L~A+96}	Ancho adicional para letra gruesa.
INT{L~A+98}	Offset de letra cursiva derecha.
INT{L~A+100}	Offset de letra cursiva izquierda.
INT{L~A+102}	Flag de ampliación.
INT{L~A+104}	Angulo de rotación del texto.
INT{L~A+106}	Color del texto.
{L~A+108}	Señalador sobre buffer para efectos del texto.
INT{L~A+112}	Offset para un segundo buffer de este tipo.
INT{L~A+114}	Color del fondo del texto.
INT{L~A+116}	Flag para copy raster form, <>0 para transparente.
{L~A+118}	Señalador sobre una rutina, que hace posible detener el proceso de relleno, en 3.0 modificado por Shift-Alternate-Control.

Tabla de los parámetros de ingreso para V_OPN(v)WK

Salvo x opcional, luego los siguientes valores default:

x Número de identificación del aparato (estándar0

1 ..: Pantalla 11..: Plotter

```
    21 ..: Impresora          31..: Metafile
    41 ..: Cámara             51..: Tabla de gráficos

1       Tipo de línea.
1       Color de la línea.
1       Tipo de marcación .
1       Color de marcación.
1       Estilo del texto.
1       Color del texto.
1       Tipo de relleno.
1       Estilo de relleno.
1       Color de relleno.
2       Sistema de coordenadas (0: NDC, 1: reservado, 2: RC).
```

Tabla del campo de WORK_OUT del VDI

Con V_OPN(v)WK están los valores de los resultados en INTOUT(0) a INTOUT(44) y en PTSOUT(0) a PTSOUT(11).

WORK_OUT(0)	Ancho máximo en pixel.
WORK_OUT(1)	Altura máxima de la imagen en pixel.
WORK_OUT(2)	0: Es posible la escala exacta de la imágen, 1: no es posible.
WORK_OUT(3)	Ancho de un pixel en micrómetros.
WORK_OUT(4)	Altura de un pixel en micrómetros.
WORK_OUT(5)	Cantidad de alturas de los caracteres de letras (0: se puede modificar).
WORK_OUT(6)	Cantidad de tipos de líneas.
WORK_OUT(7)	Cantidad de anchos de líneas (0: se puede modificar).
WORK_OUT(8)	Cantidad de símbolos de marcación.
WORK_OUT(9)	Cantidad de tamaños de los símbolos de marcación (0: se puede modificar).
WORK_OUT(10)	Cantidad de frases de caracteres.
WORK_OUT(11)	Cantidad de diseños.
WORK_OUT(12)	Cantidad de diseños de sombreado.
WORK_OUT(13)	Cantidad de colores predefinidos.
WORK_OUT(14)	Cantidad de funciones básicas para gráficos (GDP).
WORK_OUT(15)	hasta
WORK_OUT(24)	Lista de funciones básicas para gráficos (GDP).Se sostienen diez funciones básicas: 1: bar 2:arc 3:pie 4: circle 5: ellipse 6: elliptical arc 7: elliptical pie 8: rounded rectangle 9: filled rounded rectangle 10: justified graphic text El final de esta lista está marcado con -1.
WORK_OUT(25)	hasta
WORK_OUT(34)	Lista de atributos de las funciones básicas para gráficos: 0: línea 1:marca 2:texto 3: zona rellenada 4:sin atributo
WORK_OUT(35)	0: los colores no se pueden representar, 1: se pueden representar.
WORK_OUT(36)	0: El texto no se puede rotar, 1; se puede rotar.
WORK_OUT(37)	0: no son posibles funciones de relleno, 1: son posibles.

WORK_OUT(38)	0: CELLARRAY disponible, 1: no disponible.
WORK_OUT(39)	Cantidad de colores a representar: 0: más de 32767, 1: monocromático, mayor que 2 : cantidad de colores.
WORK_OUT(40)	1: Posiciona el cursor de gráficos solo con el teclado, 2: con teclado y ratón.
WORK_OUT(41)	Aparato de ingreso de valores: 1:Teclado, 2: otro aparato.
WORK_OUT(42)	Teclas de selección: 1: teclas de función, 2: otras.
WORK_OUT(43)	Cantidad de aparatos de ingreso de strings: 1: teclado.
WORK_OUT(44)	Tipo de estación de trabajo: 0: solo salida, 1:solo ingreso, 2: ingreso/salida, 3: reservado, 4: metafile.
WORK_OUT(45)	Ancho mínimo del carácter.
WORK_OUT(46)	Altura mínima del carácter.
WORK_OUT(47)	Ancho máximo del carácter.
WORK_OUT(48)	Altura máxima del carácter.
WORK_OUT(49)	Ancho mínimo visible de la línea.
WORK_OUT(50)	Reservado, siempre 0.
WORK_OUT(51)	Ancho de línea máximo en dirección x.
WORK_OUT(52)	Reservado, siempre 0.
WORK_OUT(53)	Ancho mínimo de marcación.
WORK_OUT(54)	Altura mínima de marcación.
WORK_OUT(55)	Ancho máximo de marcación.
WORK_OUT(56)	Altura máxima de marcación.

Tabla VT52

Las rutinas del VT 52-emulator se pueden llamar evidenciando el string de la siguiente tabla a través de PRINT.

CHR$(27)+"A";	Cursor para arriba (se detiene en el borde superior).
CHR$(27)+"B";	Cursor para abajo (se detiene en el borde inferior).
CHR$(27)+"C";	Cursor derecha (se detiene en el borde derecho).
CHR$(27)+"D";	Cursor izquierda (se detiene en el borde izquierdo).
CHR$(27)+"E";	Borrar pantalla (CLS).
CHR$(27)+"H";	Cursor al extremo izquierdo superior (LOCATE 1,1).
CHR$(27)+"I";	Cursor para arriba (scroll en el borde superior).
CHR$(27)+"J";	Borra la pantalla a partir de la posición del cursor.
CHR$(27)+"K";	Borra línea a partir de la posición del cursor.
CHR$(27)+"L";	Inserta línea en blanco en la posición del cursor.
CHR$(27)+"M";	Borra línea en la posición del cursor (el resto es arrastrado para arriba).
CHR$(27)+"Y"+CHR$(s+32)+CHR(z+32);	Equivale a LOCATE z,s.
CHR$(27)+"b"+CHR$(f);	Elige f como color de escritura.
CHR$(27)+"d";	Borra pantalla hasta la posición del cursor .
CHR$(27)+"e";	Conectar cursor.
CHR$(27)+"f";	Desconectar cursor.
CHR$(27)+"j";	Almacenar la posición del cursor.
CHR$(27)+"k";	Fijar cursor en la posición almacenada de CHR$(27)+"j".
CHR$(27)+"l";	Borrar línea en que se encuentra el cursor.
CHR$(27)+"o";	Borrar línea hasta la posición del cursor.
CHR$(27)+"p";	Conectar letra invertida.
CHR$(27)+"q";	Desconectar letra invertida.
CHR$(27)+"v";	Conectar el pasaje automático de líneas.
CHR$(27)+"w";	Desconectar el pasaje automático de líneas.

Mensajes de error del GFA-BASIC

0 División por cero
1 Desbordamiento
2 Número no enteroI-2147483648 .. 2147483647
3 Número no byteI0 .. 255
4 Número no palabraI-32768 .. 32767
5 Raíz cuadrada solo para I números positivos
6 Logaritmos solo para Inúmeros mayores que cero
7 Error desconocido
8 Memoria llena
9 Función o instrucción I aún no es posible
10 String demasiado largoImax. 32767 caracteres
11 No es programa en GFA-BASIC-3.00
12 Programa demasiado largoI memoria completa I NEW
13 No es programa en GFA-BASICIEOF - NEW
14 Campo dimensionado dos veces
15 Campo no dimensionado
16 Indice del campo demasiado grande
17 Dim demasiado grande
18 Cantidad errónea de índices
19 No se encontró el Procedure
20 No se encontró el Label
21 Solo permite en Open:I"I"nput "O"utput "R"andomI"A"ppend "U"pdate
22 File ya abierto
23 File #erróneo
24 File no abierto
25 Ingreso erróneo, no es un número
26 Se alcanzó el final del fileIEOF
27 Demasiados puntos para I polyline/Polyfill I máximo 128
28 El campo debe ser unidimensional
29 Cantidad de puntos mayor que el campo
30 Merge - Ningún ASCII-File
31 Merge - línea demasiado larga - detención
32 ==> Sintaxis incorrecta I detención del programa
33 Marca no definida
34 Muy pocos Data
35 Dat no numéricos
37 Diskette completo
38 Instrucción en modo directo I no es posible GOSUB
40 CLEAR no es posible en I loops For-Next o en I Procedures
41 Cont no es posible
42 Muy pocos parámetros
43 Expresión demasiado compleja
44 Función no definida
45 Demasiados parámetros
46 Parámetros erróneos I ningún número
47 Parámetros erróneos I ningún string
48 Open "R" - longitud errónea de la oración
49 Demasiados "R"-files (max.31)
50 Ningún "R"-file
52 Campos más largos quelongitud de la oración
54 GET/PUT Field-String Ilongitud errónea
55 GET/PUT número de oración erróneo
60 Longitud Sprite-String errónea
61 Error en RESERVE
62 Menú erróneo
63 Reserve errónea

64 Pointer erróneo
65 Tamaño del campo <256
66 Pointer erróneo
67 ASIN/ACOS erróneo
68 Tipo de VAR erróneo
69 ENDFUNC sin RETURN
71 Indice demasiado grande
90 Error en Local
91 Error en For
92 Resume (next) no es posible I Fatal, For o Local
93 Error de pila

Mensajes de error bomba

100 GFA-BASIC Versión 3.00I Copyright 1986-1988IGFA Systemtechnik GmbH
102 2 bombas - Bus ErrorIPeek/Poke erróneo?
103 3 nombas - Adress errorI dirección impar de palabras!I
Dpoke/dpeek, Lpoke/Lpeek?
104 4 bombas - Illegal Instruction I Instrucción de máquina no válido
105 5 bombas - Divide byZero I 68000 División por cero
106 6 bombas - CHK-Exception I 68000 instrucción CHK
107 7 bombas - TRAPV-Exception I 68000 instrucción TRAPV
108 8 bombas - Privilege Violation I 68000 violación de privilegio
109 9 bombas - Trace Exception I 68000 Trace sin Monitor

Mensajes de error TOS

-1 * Error general
-2 * Drive not ready I Se traspasó el tiempo
-3 * Instrucción desconocida
-4 * Error CRC I Suma de prueba del diskette erróneo
-5 * Bad request I Instrucción no válida
-6 * Seek error I No se halló la traza
-7 * Unknown Media I Sector boot erróneo
-8 * No se halló el sector
-9 * Sin papel
-10 * Error de escritura
-11 * Error de lectura
-12 * Error general 12
-13 * Diskette protegido, no se puede escribir
-14 * Se cambió el diskette
-15 * Aparato desconocido
-16 * Bad sector (Verify)
-17 * Insertar otro diskette
-32 * Número de función no válido
-33 * No se halló el archivo
-34 * No se halló el nombre del camino
-35 * Demasiados archivos abiertos
-36 * Acceso no es posible
-37 * Handle no válido
-39 * Memoria completa
-40 * Dirección del bloque de memoria no válida
-46 * Designación errónea de la disquetera
-49 * No hay otros datos
-64 * Error de zona GEMDOS I Seek falso?
-65 * Error interno de GEMDOS
-66 * No es programa binario
-67 * Error del bloque de la memoria