Vertel

Versión 3.00 ß5
Por Vertyos

Traducido a Español por Raúl Bores Monjiote (izerw)

Utilización de la librería

Sintaxis básica

Para utilizar a Vertel, basta con llamarlo, desde la pantalla HOME o un programa en Ti-Basic, como un programa clásico: **vertel()**
Sin embargo esta llamada no tendrá ningún efecto, sin poner nada entre los paréntesis. Los efectos dependerán de lo que ponen como valores entre los paréntesis, en forma de un nombre de función y sus argumentos.
Para las funciones que tienen argumentos, el principio es el mismo, y basta con añadir tras la función sus argumentos separados por comas. La función "**pixl**" tiene como efecto dibujar un pixel a la pantalla, y necesita de al menos dos argumentos: los datos **x**, **y** del pixel que deben indicarse.
El hecho de entrar:

vertel("**pixl**",20,20)

Dibujará pues un pixel a los datos (20,20).

Es posible entrar varias funciones sin tener que repetir varias veces la instrucción Vertel, lo que ofrece una ganancia de lugar y velocidad. Para eso, separan las funciones con una cadena vacía: **""**.
Para dibujar un pixel muy en arriba a la izquierda de la pantalla luego esperar un apoyo sobre una tecla (usando **paus**), es pues posible entrar:

vertel("**pixl**",0,0,"","**paus**").

Extensiones de la sintaxis

La sintaxis de los argumentos es bastante flexible, pues acepta algunas variaciones más o menos útiles. Si su fuente está destinada a ser leída por otros, evite no obstante utilizar un sintaxis difícil de leer, si no, aporta ninguna ventaja.
En primer lugar, si el orden de los argumentos del mismo tipo entre ellos es importante, no es el caso de los argumentos de tipos diferentes (números y cadenas). En otros términos, pueden entrar todos los argumentos numéricos en primero, o en último, poco importa puesto que permanecen en el mismo orden entre ellos:

vertel("**a**",1,"**b**",2), vertel ("**a**","**b**",1,2) y vertel(1,2,"**a**","**b**")

Son tres sintaxis exactamente equivalentes." Tenga en cuenta que eso es tan válido para los nombres de funciones, que son cadenas: pueden colocar los argumentos numéricos de una función antes de indicar su nombre, pero eso no tiene ningún interés si no dificultar su fuente a leer.

En segundo lugar, todos los argumentos numéricos omitidos son fijados en 0 por defecto. Así:

vertel("**func**",0)

Es equivalente exactamente a:

vertel("**func**")

Atención, no obstante, el 0 puede ser necesario si otros argumentos numéricos no nulos deben seguir: en el caso de:

vertel("**func**",0,1)

Es imposible suprimir del 0.

Listas en argumento

La utilización correcta de las listas en el argumento es una de las claves para realizar programas en gran medida más rápidos.
El hecho de colocar una lista en argumento en la función para cada uno de los elementos de la lista.
He aquí un ejemplo simple:

vertel("**pixl**",{10,20,30},20)

La función "**pixl**" va entonces a trazar 3 pixeles, a los datos (10,20), luego (20,20) y finalmente (30,20).
Pueden utilizar varias listas, reanudando el mismo ejemplo:

vertel("**pixl**",{10,20,30},{50,40,30})

Se dibujarán 3 pixeles también, a los datos (10,50), (20,40) y (30,30).
Si una de las listas se detiene antes de los otros, Vertel para la lectura e ignora los elementos suplementarios de las listas más largas.

Lectura ininterrumpida: Tag "|"

Por defecto, Vertel decide la lectura de las listas si se termina una de entre ellas. Los elementos suplementarios de las otras listas no se leerán pues, en tal llamada:

vertel("**pixl**",{10,20},{10,20,30})

El "30" de la segunda lista se ignora puesto que la primera lista solo tiene dos elementos y que Vertel decide la lectura.
La lectura ininterrumpida consiste precisamente en seguir la lectura aunque se terminen uno o más listas: Al colocar el "**|**" delante de una lista, se da por ininterruptible. El último elemento de la lista entonces repite tantas veces sea necesario para llenar los elementos "que faltan" con relación a las otras listas.

Ejemplo:

vertel("**pixl**", "**|**",{10,20},{10,20,30})

Se dibujarán 3 pixeles (10,10), (20,20) y (20,30): Vertel relee el último elemento "20" de la primera lista (que está en modo ininterruptible) para seguir con la segunda lista, más larga."
Atención, el tag solo afecta a la lista que la sigue. Si quieren colocar varias listas ininterruptibles, es necesario poner un tag "|" delante de cada una ella."

Lectura Secuencial: Tag "→" (flecha STO►)

Para activar la lectura secuencial, es necesario colocar el "→" antes de la lista que debe ser secuenciada, y no se considerará el tag como un argumento.
La utilización de las listas es, por defecto, simultánea. Eso significa que si se colocan varias listas en argumento, sus primeros elementos se leerán al primer ciclo, luego sus segundos, etc... en otros términos las listas "avanzan al mismo tiempo".
La lectura secuencial avanza recombinando los diferentes elementos de la lista.

He aquí un ejemplo para explicar la diferencia entre lectura simultánea y lectura secuencial:

1. vertel({1,2},{3,4})

2. vertel({1,2}, "→", {3, 4})

(Estos ejemplos solo son para la ilustración, dado que no tienen ningún efecto)

El primer ejemplo quedará (1,3) y luego (2,4), mientras que el segundo ejemplo queda: (1,3), luego (1,4), luego (2,3), y finalmente (2,4). en efecto, la segunda lista se señala como secuencial con relación a la primera, en la que solo avanza a cada ciclo de forma simultánea.
La primera lista no avanza hasta que se termina el segunda.
La lectura secuencial puede servir en varias situaciones. Por ejemplo para llenar la pantalla de una serie de PIC: al indicar los datos **x** en una lista, los datos **y** en otro, una imagen puede indicarse a todas las posiciones de la pantalla.

Es posible utilizar listas secuenciales múltiples: por ejemplo en la llamada:

vertel({1,2}, "→", {3, 4}, "→", {5, 6})

Atención, cualquier cadena que comienza por "→" se considerarán como un tag de lectura secuencial. Para evitar eso, empiece la cadena con "/": este carácter no se tomará en cuenta en la cadena, pero no lo confundirá con un tag de lectura secuencial.

Indirección internas (Direccionamiento indirecto): Tag "#"

Se trata de otra astucia muy útil, pero también puede ser considerada la más compleja.

En vez de pasar el valor de una variable en argumento de una función, por ejemplo al llamar vertel (var), pueden utilizar una indirección interna: vertel("#var")." Esto tendrá alrededor el mismo efecto: en el momento de leer la cadena, Vertel verá el # e intentará abrir la variable, para recuperar el contenido. El hecho de utilizar las indirecciones internas tiene dos consecuencias principales: hacer la llamada, y retrasar la evaluación.
La aceleración se constata en realidad si pasan grandes listas en argumento; al ejecutar por ejemplo:

: {1,2...,199,200} → list
: vertel(list)

La calculadora evalúa la lista "list" una segunda vez en la llamada, lo que toma un tiempo Al utilizar una indirección interna, con:

vertel("#list")

La calculadora no evalúa la lista, solo Vertel se encarga, de ahí un ahorro de tiempo apreciable.
El retraso de evaluación es otra consecuencia que puede resultar útil; al llamar:

vertel ("**fnc1**",var1,"","**fnc2**",var2)

La calculadora evalúa var1 y var2 en la llamada y pasa a Vertel los valores que contienen.
Sustituyendo a esta llamada por éste:

vertel("**fnc1**","#var1","","**fnc2**","#var2")

La calculadora no evalúan las variables, solo Vertel lee el contenido de las variables, sino solo lo hacen en el momento ejecutar de las funciones correspondientes. Así pues, si "fnc1" modifica el contenido de "var2", la llamada clásica sin indirecciones no tendrá en cuenta, mientras que el otro si.
Último detalle: pueden gracias a las indirecciones internas, imbricar listas, por ejemplo:

: {2, "#b"} → a
: {3,4} → b
: vertel({1,"#a"}

Se trata como 1, luego 2, 3 y 4.
Atención, de la misma forma que para otros tags, cualquier cadena que comienza por "#" se considerará como una indirección interna. Para evitar eso, empiece la cadena con "/": este carácter no se tomará en cuenta en la cadena, pero no lo confundirá con un tag de indirección interna.

Lista y descripción de las funciones

Convenios

Con el fin de simplificar la presentación, la lista de las funciones y de sus argumentos debe incluirse de la siguiente forma:

- El nombre de la función se indica en **color azul**
- Los argumentos se presentan después de los dos apartados **::**
- Entre comillas **" "** son cadenas de caracteres (string)
- Sin las comillas son números enteros positivos o negativos
- Los argumentos entre ganchos **[ ]** son opcionales
- Los argumentos entre corchetes **{/}** deben incluirse como "o uno, o otro".

Una función presentada como: **func ::** **"***var***"**, *tipo*, *[modo]* se llaman "función", y deben ir seguidas de una cadena de caracteres, de un número, y de un segundo número opcional.

Los argumentos entre **[ ]**, por lo general es el *modo*, en el manual original se ilustra de la siguiente forma:

func :: *x,y,rx,ry[**,**modo]*

y en éste manual se muestra de la siguiente forma:

func :: *x,y,rx,ry**,**[modo]*

A lo que se da entender que la coma "**,**" es también opcional, pudiendo omitir completamente éste argumento. Pero en éste manual, para evitar errores de interpretación se colocará como se menciona.

Funciones Gráficas

bufr :: *sin argumentos*

Esta función permite utilizar un buffer, o pantalla virtual, a los usos múltiples. Puede a la vez servir para efectuar operaciones gráficos invisibles para el usuario (visualización y protección), pero también y sobre todo para llamar varias instrucciones gráficos y para indicar su resultado en una única vez. La primera llamada de la función crea la pantalla virtual y repite todas las funciones de visualización sobre él. Al aparecer una segunda vez la función, el buffer se volverá a copiar a la pantalla. Tenga en cuenta que la única manera de conocer el contenido del buffers entre dos llamadas de Vertel es guardarlo (véase funciones "**pict**" y "**save**").

clrs :: *sin argumentos*

Efectúa un ClearScreen, es decir borran completamente la pantalla. Esta función difiere de la del Ti-Basic en varios puntos: Las funciones ClrIO, ClrGraph y ClrHome solo borra el contenido en ésta en cambio la función de Vertel borra o desactiva todos los pixeles en pantalla sin importar en que pantalla se encuentre (Home, I/O, Graph...); el borrado afecta a toda la pantalla y no deja ni la barra herramientas ni el statusline en la parte baja.

Ejemplo:

vertel("**clrs**")

crcp :: *x,y,rx,ry,[modo]*

Dibuja un círculo lleno; tiene por centro x, y. Se trata más exactamente de una elipse, cuyo rayo del medioeje horizontal es rx y el del vertical ry.
El modo es 0 para un círculo negro, 1 para un círculo blanco, y 2 para un círculo invertido.

crcv :: *x,y,rx,ry,[modo]*

Idéntica a la función "**crcp**", pero para un círculo vacío. Los argumentos son idénticos.

line :: *x1,y1,x2,y2,[modo]*

Traza una línea entre las posiciones x1, y2 y x2, y2. Poco importa si el primer punto es superior, inferior, a la izquierda o a la derecha del primero.
Los modos posibles son 0 para una línea negra, 1 para una línea blanca, 2 para una línea invertida, y 3 para una línea a grosor doble.

pict :: *"var",x,y ,[modo]*

Indica la imagen de tipo PIC "var" en la posición x, y. Es posible indicar la imagen fuera de la pantalla: la función se trunca, pero debido a un bug de la calculadora, el truncamiento (clipping) funciona mal con imágenes de grandes dimensiones.
Los modos son 0 para indicar normalmente en "O" lógico (como rclpic), 1 para aplastar el fondo (como rplcpic), 2 para el negativo, 3 para indicar en invertido, "O" exclusivo (como xorpic), y 4 para indicar en "Y" lógico (como andpic).

Si el modo vale -1, en vez de indicar la imagen, la función devuelve sus dimensiones en anchura y altura.

pixl :: *x,y,[modo]*

Esta función tiene una duplicación, como "**pict**". Puede modificar el estado de un pixel en los datos x, y: enciende si el modo vale 0, apaga si el modo vale 1, o invierte si el modo vale 2. Si el modo vale -1, la función no modifica nada pero devuelve el estado del pixel situado a la posición x, y sobre la pantalla (0 si está apagado, 1 si está encendido).

recp :: *x1,y1,x2,y2,[modo]*

Traza un rectángulo lleno, cuyas esquinas superior izquierdo e inferior derecho se sitúan respectivamente en las posiciones (x1, y1) y (x2, y2). Atención, la función no indicará nada si se invierten las esquinas. Los modos son 0 para indicar el rectángulo en negro, 1 para indicarlo en blanco, y 2 para indicarlo en invertido.

recv :: *x1,y1,x2,y2,[modo]*

Traza un rectángulo vacío, entre las esquinas x1, y1 y x2, y2. Atención, "**recv**" tiene un funcionamiento particular para los modos. Varios valores están disponibles para distintos efectos, y es posible combinarlos.
Si el modo vale 0 (y en consecuencia si se omite), esta función no indicará nada contrariamente a todos los otros que indican por defecto en modo normal (o en negro).
Los modos acumulables son: 0 (blanco), 1 (negro), 2 (invertido), 32 (esquinas redondeado), 64 (grosor doble), 128 (esquinas superiores cortados). Un modo a 97.1.32.64) indicará pues un rectángulo negro, de grosor doble, y con las esquinas redondeadas.

save :: *"var",x1,y1,x2,y2*

Guarda la porción de pantalla incluida entre las esquinas x1, y1 y x2, y2 en la variable "var" que será de tipo PIC.
Tenga en cuenta como se utiliza la función "**recp**" para ubicar los puntos en las esquinas.

trip :: x1,y1,x2,y2,x3,y3,[modo]

Traza un triangulo lleno con esquinas x1, y1, x2, y2, et x3, y3.
Los modos posibles son: 0 para indicar el triángulo en negro, 1 para indicarlo en blanco, 2 para invertirlo, 3 para indicarlo en líneas verticales, 4 en líneas horizontales, 5 en líneas oblicuas descendentes, y 6 en líneas oblicuas ascendentes.

Funciones Ejecutables

offp :: *sin argumentos*

Esta función apaga simplemente la calculadora. Al utilizar esta instrucción en un programa, al próximo encendido reanudará su ejecución allí donde se quedo. Tenga en cuenta también que apagar la calculadora reinicializa algunos parámetros como el plazo y la frecuencia de repetición de las teclas.

paus :: *sin argumentos*

Esta función es comparable a "pause", el programa realiza una pausa mientras el usuario no haya presionado ninguna tecla.
Posee sin embargo algunas ventajas: permite hacer una pausa en medio de una llamada de Vertel, sin dejarla y devuelve el valor de la tecla presionada (los códigos son los mismos que la función "getkey", excepto para las flechas).

retr :: *"var" ,[modo]*

Algunas funciones devuelven valores (como en el caso de “**paus**”), "**retr**" es la función que permite recuperarlos en uno o más variables. Se colocarán así todos los valores memorizados en Vertel en una variable de tipo LIST, de la primera al último.

Por ejemplo la llamada:

vertel("**paus**","","**retr**","key1")

Espera que se presione una tecla (véase función "**paus**") y devuelve el valor de la tecla presionada en la variable "key1". Si el usuario presionó la tecla ENTER (valor 13), la variable "key1" contendrá {13}.
Si el modo es diferente de 0, en vez de devolver una lista de todas los valores, Vertel devuelve solamente un único valor en una variable simple (de tipo EXPR o STR, según el tipo del valor que debe darse la vuelta).
Por ejemplo la llamada:

vertel ("**vers**","", "**paus**","","**retr**",{"var1","var2"},1)

Si el usuario apoya en ESC (valor 264) en el momento de "**paus**", colocará "3.00 ss5" en la variable "var1" (la función **vers** devuelve la versión del Vertel) y 264 en la variable "var2" (se devuelve dos veces sin interrupción el valor de fila 1: en efecto después de haber devuelto el "3.00 ss5" en "var1", se borra este valor de la memoria interna, y las siguientes se desplaza, el nuevo valor en la primera posición es pues 264).

time :: *[tiempo]*

Para medir duraciones, "**time**" utiliza un contador interno de la calculadora que tiene una frecuencia de cerca de 19 ciclos por segundo. Si el argumento *tiempo* se fija indicando un valor, el contador se inicializa a este valor. Decrementa a continuación 19 veces por segundo, hasta volver a caer a 0.
El hecho de llamar la función sin indicar *tiempo* devuelve el valor corriente del contador, permitiendo así por una simple sustracción calcular duraciones bastantes precisas.

vers :: *sin argumentos* **Esta función fue probada, y NO funciona**

Da la vuelta la versión de Vertel, o sea actualmente "3.00 ss5". Esto permite por ejemplo probar la versión de Vertel antes de ejecutar un programa, con el fin de evitar las incompatibilidades con otras versiones.
Esta porción de código puede servir de ejemplo:

```
Local ver
vertel("vers","","retr","ver",1)
 Try
   mid(ver,1)
 Else
   ""→ver
 EndTry
If ver ≠ "3.00 ß5" Then
   Text "Error : Versión de Vertel incompatible"
   Stop
EndIf
```

wait :: *tiempo*

Pone el programa en espera durante un *tiempo* (la unidad es la misma que para la función "time") 19 décimas de segundo. Se desactiva la espera si es 0 (cero), que corresponde a una duración infinita.

Funciones de Ajustes

apdt :: *tiempo*

Cambian el valor del APD Timer (*Automatic Power Down*), que es el tiempo que espera la calculadora para apagarse automáticamente al existir inactividad. La unidad del tiempo está en 19 décimas de segundo, como las funciones "time" y "wait". Si el tiempo vale 0, se desactiva el APP.

brek :: *modo*

Al ejecutar ésta función desactivan la posibilidad de causar una "interrupción" en el programa, al presionar la tecla ON. Esta posibilidad se desactivará si el modo vale 0 y activado si el modo vale 1.
Atención, el efecto es a menudo de corta duración ya que es cancelado por una gran cantidad de funciones Ti- BASIC, como, por ejemplo: **string()**

clip :: *x1,y1,x2,y2*

Cambia la zona de truncamiento (clipping), que afecta casi todas las funciones gráficas. Esta zona corresponde en realidad a la porción de pantalla (por defecto, la pantalla completa) en la cual las funciones gráficas son visibles.
Atención, la zona de truncamiento es reinicializada en cuanto la llamada de Vertel finalice.

cntr :: *[modo]*

Si el modo se omite, la función devuelve el valor del contraste (un valor entre 0 y 31). Si modo vale 0, baja el contraste de un tono, y si vale 1 lo aumenta en un tono.

keyi :: *tiempo*

Cambia el tiempo antes de la inicialización de la repetición de las teclas (por defecto: 336). Atención, el valor por defecto se restablecerán en cuanto se apague la calculadora.

keyr :: *frecuencia*

Cambia la frecuencia de repetición de las teclas (por defecto: 36).
Atención, como para "**keyi**", el valor por defecto se restablecerá en cuanto se apague la calculadora.

locl :: *nivel*

Permiten leer y modificar las variables locales de los programas "principales"; en pocas palabras, desde un subprograma (del programa principal) nosotros podemos leer y modificar directamente una variable Local que está situada en el programa principal.
Si el nivel es 1, por ejemplo, Vertel intentará encontrar las variables requeridas (con indirecciones internas, por ejemplo) un nivel atrás con relación al programa que se ejecuta la función.
Ejemplo: Si consta de solo 2 programas (programa principal y subprograma) y ponemos la función en el subprograma, al tener un nivel de 1, nosotros encontraremos la variable requerida en e l programa principal.
Lea esta porción de código para entender el funcionamiento:

```
Local sub,var
Define sub()=Prgm
Local n
Input "Valor ?",n
vertel("load",n,"","locl",1,"","retr","var",1)
EndPrgm
sub()
Pause "Valor : "&string(var)
```

La función “**load**” (ésta función se verá mas adelante) agarra el valor de “**n**” y lo introduce a la siguiente instrucción de la secuencia del Vertel, en éste caso la función “**retr**” se encarga de tomar éste valor (el de “**n**”) y guardarlo en la variable “**var**”, gracias a la función “**locl**” la variable “**var**” no es del subprograma, si no es tomada del programa principal, pero en el subprograma la trabaja como si fuera original de ésta.

Funciones de Texto

prty :: "expr",x,y,[modo]

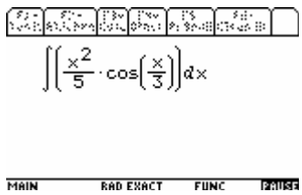
Imprime una expresión matemática "expr" a la posición x, y realizando la impresión en PrettyPrint, osea que la calculadora no evalúa la expresión y se imprime tal y cual es introducida, esta herramienta es útil si queremos mostrar los pasos intermedios de un proceso, y no queremos que la calculadora reduzca o resuelva la expresión dada.

Atención, "expr" es una cadena (string)
Procure hacer una cadena, que al ser convertida en expresión, sea completamente valida, osea que tenga todos su paréntesis, signos y sintaxis correspondiente (en caso de funciones)

Ejemplo:

vertel(“**prty**”,**”**∫(x^2/5*cos(x/3),x)**”**,20,20)

En nuestro programa obtendremos lo siguiente:



En caso de ser una expresión inválida, Vertel, se protegerá (ver al final del manual, para remediar éste problema)
Los modos son 0 para indicar normalmente ("OR" lógico), 1 para hundir al fondo, 2 para indicar en negativo, 3 para invertir, 4 para indicar en "AND" lógico, y 5 para indicado en punteados (como las funciones desactivadas de la calculadora).
Si el modo vale -1, en vez de indicar la expresión, la función devuelve sus dimensiones en anchura y altura.

stxt :: *"texto"*

Escribe un texto en el status line (la barra situada en la parte baja, conteniendo la carpeta activa, los modos, etc...) Este texto se escribirá inevitablemente en letra pequeña y se alineará a la izquierda.

Ejemplo:

text :: *"texto",x,y,tamaño,modo*

Escribe un texto sobre la pantalla, en la posición x, y. Es posible utilizar los 3 tamaños de letra disponibles, es decir 0 para la letra pequeña, 1 para la media, y 2 para la grande.
Los modos posibles son 0 para indicar normalmente, 1 para hundir al fondo, 2 para indicar en negativo, 3 para invertir, y 4 para indicar en punteados.

wdth :: *"textos",tamaño*

Devuelven la anchura en pixeles del texto con un determinado tamaño de letra.
Para alinear correctamente del texto es práctico utilizar esta función que indica exactamente la anchura del texto que debe inscribirse.
Las alturas de los caracteres 0, 1 y 2 son respectivamente 6, 8 y 10 pixeles.

Funciones de Variables

delt :: *"var",tipo*

Si el tipo es 0 suprime la variable "var" de la calculadora; si el tipo es 1, suprime la carpeta (así como su contenido) cuyo nombre esté designado por “var””.
Esta función suprimirá la variable aunque este oculto, bloqueado o archivado, así mismo borrará la carpeta aunque tengan elementos ocultos, bloqueados o archivados.

glue :: *"grupo",["var"]*

Permiten agrupar varios ficheros en uno sólo, con extensión "VTG". Así mismo éste fichero podrá extraerse para recuperar su contenido.
Si "var" se da, agarra los ficheros y los guarda en la variable “grupo”. (Con extensión “VTG”).
Si “grupo” no existe en la calculadora crea un nuevo archivo y agrega los ficheros. Si ya existiera la variable “grupo” y se ejecutará la función, solo se agregarán los ficheros a “grupo”. Si una variable se intenta agregar a “grupo” y tiene el mismo nombre de alguna variable guardada anteriormente en “grupo”, vertel omitirá la variable que se está tratando de agregar.
Para agregar varios ficheros a la vez, se ponen en forma de lista, osea entre corchetes **{}**

Ejemplo:

vertel("**glue**","grupo",{"var1", "var2", "var3",”etc”})

Si has agrupado los ficheros indicando la carpeta de la variable “var” cuando se extraigan los elementos, Vertel intentará colocarlos en la misma carpeta que le fue indicada anteriormente. Si no existe la carpeta, se preguntará al usuario para crearla.

Ejemplo:

vertel("**glue**", "grupo","main\var")

Si no se especifica la carpeta cuando se guarde una variable, cuando se extraiga esa variable, se guardará en la carpeta que esté por default en la calculadora en ese momento.
Si "var" no se indica, la función extrae todos los ficheros contenidos en el grupo, y lo guarda en la calculadora.

hide :: *"var",tipo,modo*

Permite ocultar o dejar visible una variable "var" si el tipo vale 0, si se desea ocultar una carpeta “var” el tipo vale 1.
Si el modo vale la 1 variable se ocultará, si vale 0 se volverá visible. Una variable ocultada no aparece ya en la pantalla de Var-Link en la calculadora.

list :: *["carpeta"],modo,[filtro]* **Esta función fue probada, y NO funciona**

Esta función permite obtener una lista de los ficheros de cierta carpeta en la calculadora.
Si el argumento "carpeta" no se da, la función devuelve la lista de todas las carpetas.
En cambio si se pone el nombre de una “carpeta”, devuelve la lista de los ficheros que contiene.
El argumento "modo" funciona de la siguiente forma:
Si vale 0 devolverá “programa” y si vale uno devolverá “carpeta/programa”
El filtro, opcional, sirve para filtrar los ficheros que debe devolver la función. Si no se da, devolverá todos los ficheros sin excepción; si se fija solo devolverá los ficheros de tipo que corresponderán. Si el filtro vale por ejemplo 45, lo que corresponde a las variables de tipo "STR".

load :: *{"str"/num}*

La función "load", por si sola no tiene ninguna utilidad, solo es útil cuando se combina con otras funciones de Vertel.
Sirve para tomar un valor (cadena o número) de alguna variable de la calculadora o de algún programa, y lo introduce a la siguiente instrucción de la secuencia del Vertel, osea el valor de la variable pasa a la siguiente función ejecutada en el vertel. Puede ser recuperable con la función "**retr**". Esto puede servir para completar resultados de otras funciones, pero tiene sobre todo otras utilidades como la de poder, en combinación con la función "**locl**", transmitir variables locales de un subprograma a sus lanzadores.
(Véase la función "**locl**").

make :: *"var",tamaño,[modo],["dato"]*

Crean una variable "var" de cierto “tamaño” en la calculadora.
Esta función puede ser muy útil conociendo la estructura de las variables.
El modo, si es de 0, la función pone en ceros todos los bytes de la variable. Es necesario saber que si "**make**" se utiliza sobre una variable ya existente, no será suprimida pero si redimensionada, el modo a 1 permite conservar en parte su contenido.

El último argumento “dato” (opcional), sirve para definir inmediatamente datos en la variable creada. Estos datos comienzan exactamente después de los bytes de importancia, se trata solamente pues del contenido de la variable.
El valor de cada byte debe ser representado por el carácter correspondiente (por ejemplo el valor 49 está representado el carácter(49), es decir, "1"), y pueden utilizar los valores que plantean problema (0, 2) con "\0", "\2" (atención se trata del “\” y no el "/"). El carácter "\" se considera como carácter de escape a un carácter especial.
Para utilizarlo al mismo tiempo que el "verdadero" carácter "\" (para representar el valor ord("\") = 92), es necesario utilizar "\ \".

Último cosa, se puede crear secuencias de la siguiente forma: "*NC" de ésta forma, repetimos N veces el carácter “C”: El valor de n, no es numerico, si no su valor del carácter. Ejemplo el carácter “2” al aplicarle; ord("2") quiere decir que creará una secuencia de 50 veces!).
He aquí por ejemplo cómo crear una variable de tipo NUM que contiene "123":

vertel("**make**","var1",5,”{~ε",)

ord(“{“)=123
(Reemplace el "~" por el carácter: char(1))

read :: *"var",byte,tamaño*

Devuelven "tamaño" de bytes de la variable "var", partiendo de "byte". Así si byte vale 0 y tamaño vale 4, la función devolverá 4 bytes. (el primer byte es 0, el último es tamaño-1)
Si tamaño vale 0, la función lee "var" hasta su último byte,

vertel("**read**", "var")

Permite leer una variable enteramente.
Atención, no existe ninguna restricción "fuera" en caso de lectura de la variable (más allá de su tamaño). Será entonces imposible saber a que corresponden los bytes dados la vuelta.

sequ :: *inicio,paso,cont* **Esta función fue probada, y NO funciona**

Devuelve una lista con una consecuencia de valores numéricos, cuya primera vale "inicio", el segundo "inicio+paso", el tercero "inicio+paso*2", el cuarto "inicio+paso*3", y así sucesivamente.
El interés de esta función es generar una lista de valores más rápidamente que con la función "seq" de Ti-Basic, y utiliza de a continuación en argumento para otras funciones.

vertel("**sequ**",0,3,10)

Esta llamada encarga en la lista de vuelta 10 valores: {0,3,6,9,12,15,18,21,24,27,30}

size :: *"var"*

Devuelven el tamaño en bytes de la variable "var". Es también posible conocer este tamaño con la función "**read**": el tamaño de la variable es igual a "256*b0+b1+2" si b0 es el byte n°0 y b1 el n°1.

stat :: *"var",tipo*

Devuelven el estado de la carpeta "var" o de la variable "var" en forma de 2 ó 3 bits.
Si tipo vale 0, la función considera que "var" es una variable. El primer bit entonces se arma una variable oculta, el segundo, archivado, y el tercero, bloqueado.

Ejemplo:

vertel(“**stat**”,”var”,0,””,”**retr**”,”a1”,1)

El resultado se guarda en “a1” y los valores que puede tomar son los siguientes dependiendo del estado de la variable:

Estado	Valor
Normal	0
Bloqueada	4
Archivada	6
Oculta Normal	1
Oculta Bloqueada	5
Oculta Archivada	7

El estado “normal” se considera a una variable sin archivar, (en la RAM)

Si tipo vale 1, la función considera que "var" es una carpeta. Si la carpeta se encuentra en estado oculta devuelve **1** y en estado “normal” devuelve **0**.

strn :: *num* **Esta función fue probada, y NO funciona**

Convierte un número en cadena, como lo haría la función "string()" con algunas diferencias: Vertel solo acepta los números enteros comprendidos entre -2147483648 y 2147483647, esta función no reactiva la interrupción (véase función "**brek**"), y sobre todo la gestión de las listas permite convertir una larga lista de números en una lista de cadena de forma individual.
Al efectuar string({1,2,3}) da la vuelta "{1,2,3}" en cambio al hacerlo con vertel devuelve: {"1", "2", "3"}.

test :: *{"str"/num},{"str"/num},[modo]*

La función "**test**" permite comparar dos números o cadenas, no devuelve nada si son diferentes, y devuelven un número si son idénticos. Este número en realidad se incrementa a cada ciclo en el caso de lista, lo que permite por ejemplo buscar rápidamente si un valor pertenece a una lista:

Ejemplo:

```
Local r
 {"aaa","bbb","ccc","aaa","bbb","ccc"}→list
 vertel("test","#list","bbb","","retr","r")
If dim(r)=0 Then
  Text "bbb no esta en la lista"
Else
  Text "bbb fue encontrado en: "&string(r)
EndIf
```

Este ejemplo indicará "bbb fue encontrado en: {2,5} ya que la función devolvió estas dos filas encontrando dos veces la cadena buscada en la lista.
El modo, si vale 0 (por defecto), prueba la igualdad entre los dos valores. En el caso de una comparación de números, es posible ponerle a 1 para probar si el primer número es inferior al segundo, a 2 para probar si el segundo es inferior al primero, o a 3 para probar si el primero es diferente del segundo. En el caso de una comparación de cadenas, el hecho de poner el modo a un valor N superior a 0 tiene como efecto no comparar más que los N primeros caracteres de cada cadena.

type :: *"var",[modo]* **Esta función fue probada, y NO funciona**

Devuelve el tipo de la variable "var" como lo habría hecho la función "getType()" con algunas diferencias: permite devolver el tipo de una gran lista de ficheros gracias a la gestión de listas, puede conocer el tipo real de los archivos "OTH" (que devuelve la calculadora con la función geType), y devuelve siempre el tipo en inglés, aunque la calculadora está en otro lenguaje.
Si el modo es de 1, Vertel devolverá solamente "OTH" para las variables de tipo personalizadas, como lo hace geType(), si no, indicará la verdadera extensión.
Los tipos existentes son: "NONE", "STR", "MAT", "LIST", "PRGM",
"FUNC", "DATA", "GDB", "PIC", "TEXT", "ASM", "OTH" Y "EXPR".
Para conservar la compatibilidad con geType(), la función devuelve "NONE" si la variable no existe.

writ :: *"var",byte,valor*

Permiten escribir un byte en una variable “var”. El “byte” de la variable "var" pues será aplastado y sustituido por "valor".
Para simplificar la escritura de una serie de bytes al utilizar la gestión de listas, se incrementa la variable VALOR.
Es pues inútil escribir vertel("writ","var",{2,3,4,5},{0,97,0,45}) para inscribir 4 valores a los bytes n°2, 3,.4 y 5, vertel("writ","var",2,{0,97,0,45}) tendrán el mismo efecto.

Ejemplo para un programa, usando “**clrs**”, “**stxt**”, “**prty**”, “**paus**”, “**retr**”.

El objetivo de éste programa es convertir una cadena, en éste caso es “**vc**” y mostrarla como una expresión en forma PrettyPrint con la función “**prty**” además ésta expresión la podemos mover con las teclas de dirección, y nos salimos con ENTER.

```
{0,0}→keyscro
1→scrollx
While keyscro[1]≠13

vertel("clrs","","stxt","Use las teclas ← → para desplazarse"
vertel("prty",vc,scrollx,35,"","paus","","retr","keyscro")

  If keyscro[1]=338 Then
 scrollx+20→scrollx
EndIf
If keyscro[1]=344 Then
 scrollx-20→scrollx
EndIf
EndWhile
```

Se puso 2 veces el vertel, para que alcanzara en la pantalla, pero se puede poner todo junto solamente separando con **,””,** como se ha mencionado anteriormente.

Recomendaciones:

El vertel al poner mal la sintaxis de los argumentos, o poner argumento inválidos, el vertel, se protegerá, quedando como una variable oculta, para solucionar el problema archive las variables de su calculadora y resetee, así volverá aparecer el vertel.

Alguno errores comunes pueden ser advertidos desde el mismo vertel, usando el archivo Vertel Developper, el cual contiene exactamente lo mismo que el archivo “normal” pero con diferencia que advierte alguno errores comunes, como por ejemplo al usar la función “**retr**” puede advertir al usuario que la variable nunca fue modificada o creada, por lo tanto uno puede deducir que la función anterior en ejecución no devolvió ningún valor.
Use éste archivo mientras diseña su programa, para la distribución y versión final de su programa use los archivos:

- Vertel.89z
- Vertel.9xz
- Vertel.v2z

(Según sea el modelo de calculadora)

Nota:

Ésta guía fue traducida por Raul Bores Monjiote (izerw), no todos los textos son traducciones fieles a la original, en su mayoría fueron redactados, según experiencia propia y haciendo algunas correcciones y adaptaciones al Español.

La mayoría de las funciones fueron probadas (para entender su funcionamiento) algunas de ellas tienen una etiqueta en rojo, explicando que fueron probadas pero no funcionan, si alguien considera que algunos de éstas funciones funcionan bien, por favor contacta al correo que se menciona abajo, para corregir lo mas pronto posible éste documento.

Cualquier duda o comentario:

Escriba a: Izerw1@gmail.com

ó visite el foro de calculadoras.cl dedicado a las TI:
http://www.calculadoras.cl/foro/forumdisplay.php?f=2