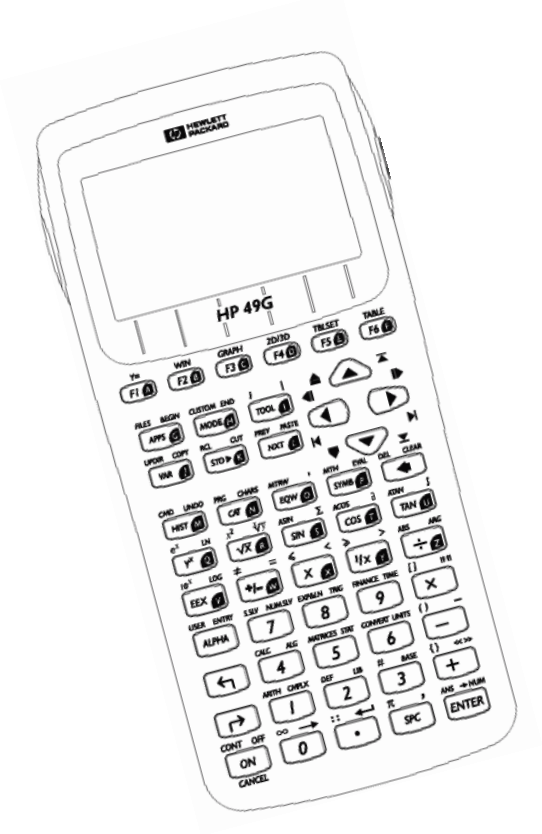
Programando

en

UserRPL

Roger G. Broncano



4ta Revisión

¡Sierra de mi Perú, Perú del mundo,
Y Perú al pie del orbe, yo me adhiero!
César Vallejo

¿Será que he nacido desarmado del todo para luchar con el mundo?
Puede ser
Pero este sobresalto diario viene a dar directamente a mi voluntad,
y la apabulla
y parece haberla tomado de presa preferida.
En medio de mis horas más horribles, es mi voluntad la que vibra,
y su movimiento va desde el punto mortal
en que uno se reduce a sólo dejar que venga la muerte,
hasta el punto en que se tienta
¡conquistar el universo entero, a sangre, y fuego!
¡Y sin embargo, es una voluntad estéril, baldada, la mía!
César Vallejo

Es posible que mañana muera,
y en la tierra no quedará nadie
que me haya comprendido por completo.
Unos me considerarán peor
y otros mejor de lo que soy.
Algunos dirán que era una buena persona;
otros, que era un canalla.
Pero las dos opiniones serán igualmente equivocadas.
Mijail Iurevitch Lérmontov

I'm not like them but I can pretend
The sun Is gone but I have a light
The day is done but I'm having fun
I think I'm dumb or maybe just happy
Think I'm just happy
Nirvana

Prólogo

How I wish, how I wish you were here
We're just two lost souls
Swimming in a fish bowl,
Year after year,
Running over the same old ground
What have we found?
The same old fears
Wish you were here
Pink Floyd

Oh, please give me a little more
And I'll push away those baby blues
Cause one of these days this will die
So will me and so will you
Blind Melon

Aún en mi mente tengo el recuerdo (quizá hasta algo nostálgico) de la vez en que un compañero de la Facultad de Ingeniería Civil me presentó la calculadora HP. No pensé que sería tan maravillosa, tampoco que me brindaría grandes satisfacciones. Aunque debo reconocer que al principio no podía hacer más que algún simple cálculo con ella y ahora escribo un manual pensado para alguien que esté dispuesto a leer y a aprender.

Han pasado muchos años desde aquella vez y definitivamente se notan los años, sobre todo en mi vieja HP48GX, ya que basta con mirarle el estuche para darse cuenta de ello. Ahora ya se han hecho mejoras a la calculadora y han salido al mercado nuevos modelos, los usuarios estamos contentos por obvias razones y es que vemos en cada nuevo modelo una expansión de posibilidades y conocimientos.

La calculadora HP posee características para la programación fabulosas, lo que nos permite virtualmente desarrollar cualquier cosa, teóricamente todas las capacidades y funciones que posee la calculadora son accesibles por el usuario y se puede hacer todo lo que la calculadora puede hacer.

Creo que se me ocurrió algo interesante en qué pensar, de que hablar y escribir. Y definitivamente espero que así sea.

Huaraz, Junio 2007
Roger G. Broncano

Agradecimientos

Ahora me resulta inevitable agradecer a todos aquellos que de alguna manera, en realidad de tantas maneras, me ayudaron a realizar este manual. En primer lugar quiero mencionar A mi abuelo *Florián* (en algún lugar), por el regalo tan especial. A mis hermanas *Mayra* y *Flor*, las dos razones que me dio la vida para respirar. A mi pequeño amigo *Jimi*, convertido ahora en la esperanza de los sueños que no viví. A mis grandes amigos: *Edwar*, *Héctor*, *Marco, Richard, Roberto, Marcos, Fernando y Daniel*, por su apoyo en mis muchos proyectos. A todas las maravillosas poseedoras del sexo femenino y de cuyas vidas fui parte alguna vez. A *Hewlett-Packard* por la calculadora. A todas las personas que se gastan la vida ayudando a crear un mundo mejor. Y finalmente a los creadores de aquellas eternas melodías que acostumbraron mis oídos, a todos aquellos a quienes les debemos el *café* y a la encantadora *Alicia* que resultó ser *Mónica*, las culpables de un insomnio colosal.

Siempre quise escribir algo que fuera de utilidad para los demás, algo de lo que me pudiera sentir satisfecho. Posiblemente algún otro día veré el fruto de mi trabajo, pero no me reconoceré en él porque serán otras personas en otro momento. Mañana ojearé con melancolía este trabajo y recordaré aquellos fragmentos ajenos que me enseñó la vida…

Dormir es distraerse del mundo
Jorge Luis Borges

Sigue llenando este minuto, de razones para respirar
No me complazcas, no te niegues, no hables por hablar
Pablo Milanés

El que tenga una canción tendrá tormenta, el que tenga compañía, soledad
el que siga buen camino tendrá sillas peligrosas que lo inviten a parar
pero vale la canción buena tormenta y la compañía vale soledad
siempre vale la agonía de la prisa aunque se llene de sillas la verdad
Silvio Rodríguez

Quizás porque, no soy un buen poeta
Puedo pedirte que te quedes quieta
Hasta que yo termine estas palabras
Sui Generis

No es querer decir amor y que la lengua no llegue,
es tener lengua y no llegar al amor
José Saramago

Tabla de Contenidos

Introducción

...a veces pienso que estoy en el lugar equivocado,
en un mundo que me es ajeno y extraño,
y cada intento por ser un rayo de sol se vuelve inútil,
entonces huyo del mundo y empiezo a soñar...

El lenguaje de programación UserRPL, es uno de los lenguajes disponibles para programar la calculadora HP, este lenguaje le permitirá al usuario aprovechar la mayoría de sus capacidades. El conocimiento de la programación en UserRPL, nos ayudará por lo tanto a sacarle el máximo provecho a la poderosa calculadora y crear desde programas sencillos hasta increíbles programas gráficos con interfase de usuario y mucho más.

El propósito de este manual es justamente ese, que puedas expandir tus conocimientos y de alguna forma aprender a programar en UserRPL. La calculadora HP es una herramienta de trabajo muy poderosa, de uso variado y flexible que la convierte en un instrumento fundamental a la hora de realizar grandes tareas. La programación constituye una pieza fundamental para aprovechar las características que le dieron sus creadores.

El Manual está dividido en 15 capítulos, los primeros capítulos se dedican un poco más al trabajo con la calculadora, el manejo fundamental de la pila, el trabajo con archivos, de tal manera que nos permitan tener una base para el aprendizaje del lenguaje RPL. En los capítulos siguientes se enseñan aspectos básicos del lenguaje y el manejo de los objetos de la calculadora, la creación de simples programas en aplicación de los comandos y estructuras de control, hasta llegar a conceptos más avanzados en donde aprenderás características especiales que dispone la calculadora para hacer que tus programas puedan ser fáciles de usar y sumamente intuitivos. Las explicaciones que se presentan en cada capítulo son complementadas con ejemplos prácticos para su mejor entendimiento.

Roger G. Broncano
Huaraz, Junio 2007

Manejo de la Pila

Mi primera taza de café y estoy pensando lo que voy a escribir…

El lugar donde la calculadora guarda los *objetos* que vas ingresando, así como los resultados de las operaciones se denomina "PILA". El manejo adecuado de la pila se hace necesario antes de realizar un programa, ya que nos da una idea de las operaciones sucesivas que ejecuta un programa y que se llevan a cabo dentro de ella.

Los *objetos* de la calculadora vienen a ser números (reales ó complejos), cadenas, matrices, listas, programas, gráficos y otros, los cuales explicaremos detalladamente en el siguiente capítulo.

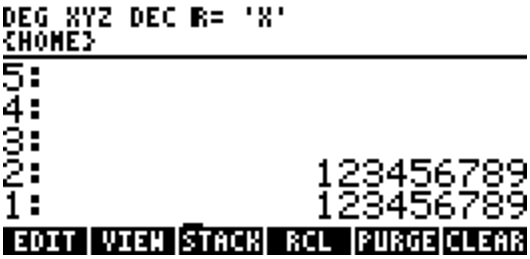
Cuando escribes cualquier *objeto*, notamos que automáticamente se inserta en el nivel 1 de la pila ó realiza operaciones (si se trata de un comando) tomando de la pila los argumentos necesarios y devolviendo resultados ó acciones. De esta manera los *objetos* contenidos dentro de la pila van ocupando un nivel (1, 2, 3, etc.), incrementándose ó disminuyendo estos de acuerdo a las operaciones que vayamos realizando.

La calculadora posee comandos para trabajar de manera eficiente con los valores que se encuentran en la pila y operar con ellos. Es el caso del comando **DUP** que se encarga de hacer una copia del objeto que se encuentra en el nivel 1 de la pila.

Ejemplo:

Antes del comando **DUP**

Después del comando **DUP**



El conocimiento de las operaciones en la pila será fundamental a la hora de realizar nuestros programas. Para no extender demasiado el manual no me detendré a explicar todos los comandos, para ello pueden consultar su manual de usuario. En el manual de usuario podrás encontrar cada uno de los comandos explicados uno a uno con sus definiciones y parámetros de uso.

La mayoría de comandos para manejar la pila se encuentran disponibles en el menú [**STACK**].

IMPORTANTE:
Cuando se inicia la edición de cualquier objeto en la calculadora, se nota que la pila se desplaza hacia arriba y se muestra una línea de edición, a ésta línea se le denomina *Línea de Comandos*. Esta línea de comandos nos servirá para hacer las entradas de objetos a la pila o para ejecutar comandos.

Los Objetos de la Calculadora

El primer paso para iniciarse en la programación es conocer adecuadamente los objetos con los que puedes trabajar en la calculadora, ya que la variedad y la flexibilidad de estos nos servirá para elegir de manera adecuada el método de trabajo al construir determinado programa. Los objetos con los que trabaja la calculadora son bastante potentes, hablamos por ejemplo de las listas que tienen una gran variedad de usos. Pasemos a ver entonces un poco acerca de los objetos que maneja la calculadora, no los vamos a explicar todos, sólo los de uso más común dentro de la programación. Si deseas saber el tipo de objeto definido dentro de la calculadora puedes usar el comando **TYPE**.

Números Reales y Complejos

Creo que aquí no hay mucho que explicar, pienso que los conoces bastante, excepto que los números complejos pueden ser usados como coordenadas ya sean cartesianas ó polares y que muchas de las funciones utilizan este tipo de objetos como argumentos que requieren coordenadas (sobre todo los comandos para gráficos).

Ejemplo:

(5,8) → esto sería el complejo 5+8i, pero también puede ser la coordenada: X=5, Y=8.

Secuencia de Caracteres

Las secuencias de caracteres son bastante flexibles y su uso se da generalmente para describir pasos en los programas, la flexibilidad de estos nos permitirá guiar al usuario en la introducción de datos.

Ejemplo:

"INGRESE EL DATO N° 1" → el delimitador para este tipo de objeto son las comillas.

Sistema Real y Complejo

Son las matrices y aunque en el modelo HP48 no son tan flexibles como en el modelo HP49, nos serán de mucha utilidad al momento de programar.

Ejemplo:

```
[ [ 1 2 ]
  [ 3 4 ] ]
```

Listas

Las listas son a mi parecer el objeto mejor elaborado de la calculadora, su flexibilidad y facilidad de uso la convertirá en el objeto con el que posiblemente tengamos que trabajar siempre, ya que tiene la capacidad de almacenar todos los objetos de la calculadora, incluidas ellas mismas. También son usadas para representar coordenadas con las funciones gráficas.

Ejemplo:

```
{ 1 "HOLA" [ 1 2 3 ] { (1,2) } }
```

Nombre Global y Local

En este caso estamos hablando de las variables, en el manual me referiré a los nombres globales y locales como variables. Son usados para representar con nombres a los objetos de la calculadora, debemos tener en cuenta que hay ciertas reglas para usar estos nombres como por ejemplo, que no pueden contener números al principio ó no pueden usarse nombres de comandos, etc. (Algún día sabrán que esto no es del todo cierto).

Ejemplo:

Los nombres de objetos que tienes almacenado en tu calculadora, como programas, directorios, listas, ecuaciones son ejemplos de *nombres globales*, los *nombres locales* se usan dentro de los programas.

Programa

Pues estos son los objetos que harán el trabajo pesado. Los programas también pueden usarse como funciones dentro de las ecuaciones, ya que son capaces de recoger y devolver argumentos. Y aquí empezaremos el tema principal del manual con el ejemplo más típico:

Ejemplo:

```
«
  "HOLA MUNDO" MSGBOX
»
```

Objeto Algebraico

En general vienen a ser las ecuaciones y son de uso regular dentro de los programas.

Ejemplo:

```
'X+Y'
```

Entero Binario

Son números especiales que maneja la calculadora, ya que poseen características que no se pueden ejecutar con los números reales como por ejemplo el cambio de bases numéricas, desplazamiento de bits, etc. También son usados por las funciones gráficas junto con las listas para representar coordenadas.

Ejemplo:

```
# 2d (número decimal), # 45FDh (número hexadecimal),
{ # 4d # 6d } (Coordenadas).
```

Objeto de Gráficos

El uso de los gráficos se da sobre todo en la gráfica de funciones, pero también vamos a usarlos para la representación de datos, resultados y los infaltables juegos.

Objeto Etiquetado

Son usados para representar un objeto con su descripción (etiqueta). Al usar la calculadora se dan cuenta por ejemplo que al momento de resolver ecuaciones hay resultados que se devuelven a la pila con un texto descriptivo, pues esos son los objetos etiquetados. Una característica de estos objetos al momento de realizar operaciones, es la posibilidad que tienen estos de mantener su valor de objeto primario, es decir, si tengo dos números reales debidamente etiquetados y deseo sumarlos lo puedo hacer sin necesidad de quitar la etiqueta de los objetos y obtendré como resultado la suma de los dos números reales.

Ejemplo:

`:BASE: 4, :ALTURA: 5`, si los sumo obtendré 9.

Otro aspecto a tomar en cuenta dentro de la programación es el de trabajar de manera ordenada y repartir el trabajo (recuerda: "*DIVIDE Y VENCERAS*"). Si se trata de proyectos grandes es mejor dividir el programa en varios módulos que te permitan editar y depurar tus programas con facilidad. Por ejemplo para proyectos grandes puedes crear un directorio donde almacenes todo el programa. Con el que tendrías el siguiente esquema para cualquier proyecto:

```
DIRECTORIO
  ├─ PROGRAMA PRINCIPAL
  ├─ MODULO 1
  └─ MODULO 2
     ...
```

Es recomendable para lograr un orden, utilizar directorios de tal manera que podamos organizar la calculadora y no tener variables dispersas por todos lados, lo cual me recuerda a mi madre, se horrorizaría si me oyera hablar de orden. También recomiendo evitar el uso de ecuaciones dentro de los programas a menos que sean necesarias, ya que son lentas y ocupan mucho más espacio. La calculadora utiliza un tipo de notación para operar denominada: Notación Polaca Inversa (RPN: *Reverse Polish Notation*), que requiere primero los argumentos y después la

función ó comando para realizar sus operaciones. Aunque en los nuevos modelos de calculadora se ha implementado el modo algebraico, el modo RPN continúa siendo a mi modesto parecer el modo más adecuado para trabajar en la calculadora y si te acostumbras a trabajar de esta manera verás lo sencillo que resulta programar. De todos modos también puedes hacerlo mediante objetos algebraicos, aunque no en todos los casos es posible.

Veamos el siguiente ejemplo:

```
«
  1 'A' STO
  2 'B' STO
  3 'C' STO
  'A^2+B^2+C^2+2*A*B*C' →NUM
»
```

Esto se vería de la siguiente manera sin utilizar ecuaciones:

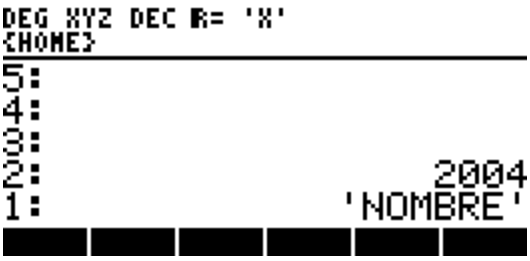
```
«
  1 'A' STO
  2 'B' STO
  3 'C' STO
  A 2 ^ B 2 ^ + C 2 ^ + 2 A * B * C * +
»
```

Las dos formas son perfectamente válidas.

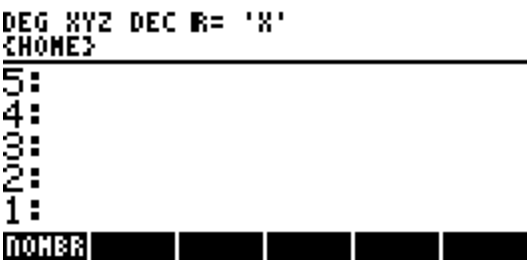
Manejo de Archivos

En la calculadora, cuando hablamos de archivos nos referimos a las *variables* que aparecen cuando presionamos la tecla [**VAR**], que no son sino *Variables Globales* que se van creando y que pueden ser nuevos ó tras el resultado de algunas operaciones como la solución de ecuaciones. Podríamos compararlas con los archivos de una PC, pero resulta que los de la calculadora tienen características especiales.

En las variables que iremos creando se puede almacenar cualquier objeto de la amplia variedad que existen en la calculadora. Para crear una nueva variable insertamos en la pila el objeto que contendrá la variable, luego el nombre de la variable:



Luego de la ejecución del comando **STO** tendremos:



Las variables siempre estarán disponibles desde el menú **[VAR]**, y podemos posteriormente editarlas para modificar su valor, reemplazarlos por otros objetos ó simplemente verlos, ésta es una gran ventaja que nos permite la calculadora.

Para borrar las variables que hemos creado debemos de insertar el nombre de la variable en el nivel 1 de la pila y utilizar el comando **PURGE**.

Para el trabajo con directorios la calculadora posee los comandos **CRDIR** para crear directorios y **PGDIR** para borrar directorios.

Transferencia de Archivos

En el capítulo anterior tratábamos acerca del trabajo con archivos. Pues bien a veces se hace necesario transferir archivos desde ó hacia otros dispositivos, por ejemplo cuando queramos utilizar programas que hemos bajado de Internet. En este capítulo veremos como transferir estos archivos usando la computadora u otra calculadora.

Transferencia Entre Dos Calculadoras

Para transferir archivos entre dos calculadoras necesitan estar conectadas mediante un cable serial ó en el caso de dos calculadoras HP48 los infrarrojos deben estar en contacto. Luego seguiremos los siguientes pasos para transferir los archivos:

- En la calculadora que contiene los archivos a transferir seleccionamos de la lista **I/O** la opción SEND TO HP4X..., escribimos el nombre de la variable ó lo seleccionamos con el menú CHOOS.

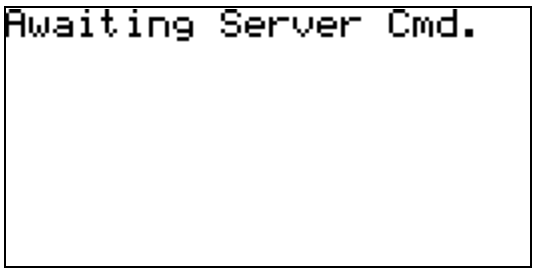
- En la otra calculadora seleccionamos la opción GET FROM HP4X de la lista **I/O**.
- Presionamos SEND en la primera calculadora y listo.

Luego de transferidos los archivos aparecen en el menú [**VAR**].

Transferencia Entre Calculadora y PC

Para transferir archivos entre una PC y nuestra calculadora ó viceversa se requiere que estén conectadas mediante un cable serial, además será necesario un programa de comunicación. Luego seguiremos los siguientes pasos:

- Debes prender tu calculadora y ponerlo en modo servidor, para ello debes presionar las teclas de [***cambio derecho***] y [***flecha derecha***].



- En la computadora, enlazamos la computadora y la calculadora con la ayuda del programa de comunicaciones, puedes utilizar el programa PC Connectivity Kit de Hewlett Packard. En el programa de comunicación debemos establecer algunas opciones como por ejemplo el puerto de comunicación (generalmente COM1), la velocidad (generalmente 9600 baudios) y por último el modo de transferencia: binario ó ASCII, se recomienda la transferencia binaria.

- Dependiendo de la facilidad de uso del programa la transferencia no te tomará sino una breve espera y podrás tener el archivo en tu calculadora ó en la computadora.

Luego de transferidos los archivos, estos aparecerán en el menú [**VAR**].

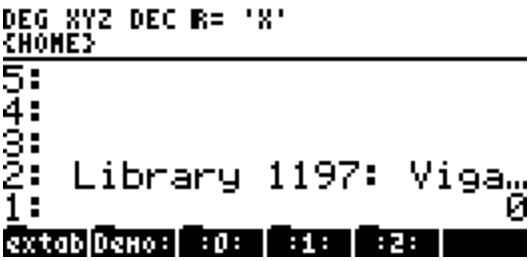
Después de la transferencia de archivos a tu calculadora, estos aparecerán con el mismo nombre de origen en el menú [**VAR**], en el caso de programas, los puedes utilizar directamente, pero, las librerías deberán ser instalarlas, para esto puedes consultar los pasos descritos en el siguiente capítulo. Luego de instalar las librerías no es necesario guardar los archivos transferidos en tu calculadora, puedes borrarlos ya que ocupan espacio en memoria.

Instalación de Programas y Librerías

Muchas veces nos encontramos con programas y librerías que conseguimos de nuestros amigos ó en Internet. En este capítulo vamos a mostrar como instalar los programas y librerías que transferimos a nuestra calculadora para poder utilizarlas.

Para poder instalar un programa, éste deberá aparecer en el nivel 1 de la pila, en un capítulo anterior aprendimos como trabajar con estas variables, entonces lo que sigue es colocar un nombre descriptivo y ejecutar el comando **STO**, para borrarlo seguimos el mismo procedimiento del mismo capítulo.

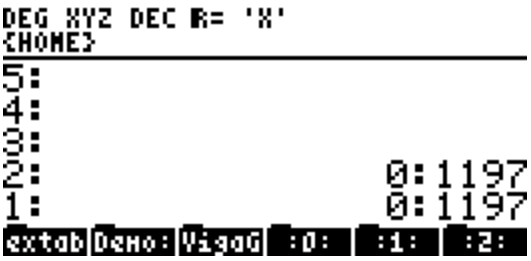
En el caso de las librerías debemos recordar que estos se guardan en un puerto lógico, no es posible su edición y están disponibles desde el menú [**LIBRARY**]. Lo que se hace en este caso es guardar la librería en un puerto (puede ser el puerto 0), para ello recuperamos el objeto librería en la pila, insertamos el número de puerto y finalmente el comando **STO** como se aprecia a continuación:



Luego de ejecutar el comando **STO**, la librería aún no está disponible, necesita instalarse, para ello apagamos y volvemos a encender la calculadora y la librería quedará instalada, podemos acceder a ella en el menú [**LIBRARY**].

En algunas ocasiones las librerías para instalarse necesitan reiniciar la calculadora, para ello basta con presionar las teclas [**ON**]-[**C**] y la librería quedará instalada.

Para desinstalar las librerías se sigue un procedimiento similar al descrito en el capítulo anterior con las variables, la diferencia es que en este caso se requiere el número de la librería en el nivel 1 de la pila etiquetado con el puerto en el que se encuentra instalado, duplicamos el argumento y aplicamos los comandos **DETACH** y **PURGE**. Como se ve a continuación:



Luego de ejecutados los comandos la librería queda desinstalada, aunque algunas veces es necesario reiniciar la calculadora cuando recibes el mensaje “ERROR: Object in Use” y volver a efectuar los pasos descritos anteriormente, ya que seguramente existen referencias a la librería que necesitan ser eliminadas, para esto es necesario el reinicio. En la calculadora HP49, puedes simplificar este procedimiento recurriendo al **File Manager**.

Variables Locales y Globales

Es hora de otra taza de café porque lo que sigue es realmente interesante.

Variables Globales

Como ya habíamos mencionado en el capítulo que trata sobre los objetos de la calculadora, la variables globales son todos los objetos que tienes almacenados en tu calculadora y las que se van a crear de manera permanente (por decirlo de alguna forma) manualmente ó por medio de los programas. Su uso dentro de los programas representa una desventaja, ya que el trabajo con estas variables es un poco lento y para borrarlas necesitas hacer uso del comando **PURGE**. En la mayoría de casos no vas a tener necesidad de crear variables globales, ya que generalmente las variables se usan de manera temporal y es aquí donde entran las variables locales. Pero veamos un ejemplo del uso de variables globales.

Ejemplo:

```
«
  6 'X' STO
  8 'Y' STO
  'X^2+Y^2' EVAL
»
```

Al ejecutar el programa notaremos que han aparecido dos nuevas variables en el directorio actual. Si queremos que las variables no aparezcan tendríamos que hacer lo siguiente:

```
«
  6 'X' STO
  8 'Y' STO
  'X^2+Y^2' EVAL
  'X' PURGE
  'Y' PURGE
»
```

Uno de los aspectos que siempre me ha preocupado es la optimización, entonces si se han fijado en el programa anterior estoy usando el comando **PURGE** 2 veces cuando podría haber hecho lo siguiente:

```
«
  6 'X' STO
  8 'Y' STO
  'X^2+Y^2' EVAL
  { X Y } PURGE
»
```

Código con el que me ahorrado algunos bytes, he hecho el programa más legible y más rápido. Para algunos no puede significar mucho, pero si sumas esas milésimas de segundo en una iteración los resultados serían obvios. Además debemos tener en cuenta que hay muchos comandos como **PURGE** que aceptan argumentos individuales ó grupales como listas.

Variables Locales

Las variables locales son aquellas variables temporales dentro de los programas, es decir solo están presentes en la ejecución de los programas y serán las variables con las que vamos a trabajar frecuentemente.

Existen ciertos pasos para crear variables locales, los que pasamos a describir a continuación:

```
«
  → A B C
  «  @ secuencias del programa
  »
»
```

En el ejemplo se muestra la secuencia para crear variables locales. El comando (→) es el encargado de recoger 3 valores de la pila y almacenarlos en las variables definidas; en este caso el nivel 3 se almacenará en A, el nivel 2 en B y el nivel 1 en C. A continuación se ha iniciado otro programa ó subprograma, esto se hará generalmente cuando se tengan programas complejos cuya secuencia de comandos sea mucho más extensa. Debemos aclarar que estas variables solo se pueden usar dentro de este subprograma y que no existen en otros entornos ó subprogramas. Por ejemplo tenemos la necesidad de crear una función tendríamos lo siguiente:

```
«
  → A B C '(A+B)/C'
»
```

Nótese que no es necesario evaluar la ecuación ya que esta se evaluará inmediatamente después de recoger los argumentos. Esta sería una forma de elaborar funciones para utilizarlas dentro de otras ecuaciones. Claro que las funciones también podrían ser mucho más complejas y simplemente debemos de hacer que el programa retorne un valor ó valores.

En el caso de las variables locales no es necesario el proceso de eliminación ya que estas se eliminan una vez terminado el programa. Además para los nombres puedes utilizar cualquier carácter, como los caracteres del alfabeto griego, algunos símbolos y números, siguiendo siempre la convención para nombres de la calculadora. Puedes utilizar un nombre lo suficientemente descriptivo para tus variables, pero recuerda que cuanto más largo sean ocupan más espacio dentro de los programas. Además de éste tipo de variables locales, existen otro tipo de variables, denominadas: *variables locales compiladas*. Su uso depende de la necesidad y requerimientos de programación, pero por lo general no serán necesarias. Las variables locales compiladas vienen a ser variables que no sólo existen dentro de un subprograma sino dentro de otro módulo que se ejecute bajo el programa principal.

Veamos el ejemplo:

```
«
  → A ←B
  «
    FUN1 A * A -
  »
»
```

En el ejemplo estamos recogiendo dos valores, una en A que es una variable local y otra en B que es una variable local compilada. Como se habrá notado la variable B posee un símbolo (←) que lo precede y es la manera como se le hará referencia en otros subprogramas. Dentro del programa se hace referencia al subprograma FUN1, la definición de éste programa sería:

```
«
  2 ←B ^ ←B /
»
```

Nótese que no hay necesidad de crear la variable B pues esta es visible desde el programa que lo llamó. Cabe recalcar además que si ejecutas sólo el programa FUN1, sin llamarlo desde otro programa recibirás el error "Undefined Local Name" ya que no existe ninguna instancia ó programa que haya creado previamente la variable.

Estructuras de Control Condicionales

Ahora me encuentro en un dilema, no sé si tomar otra taza de café o tomar un respiro porque escribir los capítulos anteriores me han dejado un tanto agotado. Pero vayamos a lo que nos interesa.

Las estructuras de control condicionales permiten que el programa tome una decisión en el transcurso de su ejecución. Estas estructuras se encuentran disponibles en el menú BRCH.

IF … THEN … END

Usado para realizar determinadas acciones si se cumple una condición.

Sintaxis:

```
«
  @ secuencias del programa
  IF @ condición
  THEN
   @ secuencias a ejecutar si se cumple la condición
  END
  @ secuencias del programa
»
```

Ejemplo:

```
«
  → a
  «
    IF 'a>0'
    THEN
      "NUMERO POSITIVO"
    END
  »
»
```

El programa tomará un valor (variable local 'a'), comprueba que el número sea mayor que 0, si lo es, entonces devolverá un mensaje, si no lo es, no devolverá ningún valor. Debemos notar que la condición puede ser algebraica ó en RPN, las condiciones algebraicas serán evaluadas por la estructura, de manera que no será necesario evaluarlas.

IF ... THEN ... ELSE ... END

Usado para realizar determinadas acciones si se cumple una condición, además de poseer la estructura correspondiente para realizar acciones, si no se cumple la condición.

Sintaxis:

```
«
  @ secuencias del programa
  IF @ condición
  THEN
   @ secuencias a ejecutar si se cumple la condición
  ELSE
   @ secuencias a ejecutar si no se cumple la condición
  END
  @ secuencias del programa
»
```

Ejemplo:

```
«
  → a
  «
    IF a 0 >
    THEN
      "NUMERO POSITIVO"
    ELSE
      "NUMERO NEGATIVO"
    END
  »
»
```

El programa tomará un valor (variable local 'a'), comprueba que el número sea mayor que 0, en cualquier caso devolverá mensajes de acuerdo al número introducido.

CASE … THEN … END … END

Usado para realizar determinadas acciones si se cumplen una serie de condiciones, su uso es alternativo a los IFs anidados.

Sintaxis:

```
«
  @ secuencias del programa
  CASE @ condición 1
  THEN
    @ secuencias a ejecutar si se cumple la condición 1
  END (condición 2)
  THEN
    @ secuencias a ejecutar si se cumple la condición 2
  END
    @ secuencias a ejecutar por defecto
  END
  @ secuencias del programa
»
```

Ejemplo:

```
«
  → a
  «
    CASE a 0 >
    THEN
      "NUMERO POSITIVO"
    END a 0 <
    THEN
      "NUMERO NEGATIVO"
    END
      "CERO"
    END
  »
»
```

El programa tomará un valor (variable local ‘a’), comprueba que el número sea mayor que 0 y devuelve un mensaje positivo, si no lo es comprueba que sea menor que cero y devuelve un mensaje negativo, si no es ninguno de los dos casos devolverá el mensaje cero. Noten que el tercer caso no necesita condición, se ejecutará si no se cumplen las anteriores, puede dejarse en blanco de no ser necesario.

Usando IFs anidados se vería de la siguiente manera:

```
«
  → a
  «
    IF a 0 >
    THEN
      "NUMERO POSITIVO"
    ELSE
      IF a 0 <
      THEN
        "NUMERO NEGATIVO"
      ELSE
        "CERO"
      END
    END
  »
»
```

EJEMPLOS DE CONDICIONES

Pero las condiciones no sólo puedes ser de este tipo, la calculadora dispone de una infinidad de maneras para realizar comprobaciones según el caso en el que nos encontremos. Podemos hacer uso de los operadores lógicos, del tipo de objeto, los FLAGS ó banderas de usuario y del sistema, el sistema dispone de 64 banderas (-1 ... -64) así como las del usuario (1 ... 64); en la calculadora HP49 estas banderas han sido ampliadas al doble. Aquí, algunos ejemplos:

Ejemplos:

RPN	Algebraico
A 0 > B 0 > AND	'A>0 AND B>0'
5 FS?	
B 8 >	'B>8'
X 5 == Y 10 == OR	'X==5 OR Y==10'
1 FC?	
V TYPE 4 ==	
C "NAME" SAME	

IFERR ... THEN ... ELSE ... END

Estas estructuras se usan para la detección de errores, es decir para que tu programa pueda controlar los errores y no salgan los mensajes de la calculadora, puede ser muy útil, aunque este proceso es un poco lento. Una buena alternativa de uso puede ser para capturar la tecla **[CANCEL]**. La sintaxis y las comprobaciones son iguales que en los casos anteriores.

Ejemplo:

```
«
  IFERR
    "DIRECTORIOS:" 15 TVARS 1 CHOOSE
  THEN
    3 DROPN
  ELSE
    DROP EVAL
  END
»
```

El programa de ejemplo muestra los directorios en la ruta actual, pero ocurrirá un error si no existen directorios, en tal caso borramos los argumentos y no se muestra nada, pero si existen, mostrará los directorios en un cuadro de selección en donde si elijes algún nombre, te cambiarás a ese directorio.

Estructuras de Control de Bucle

Me tomo un respiro (se me acabó el café), mientras vuelvo a escuchar por n-ésima vez la canción “The Promise” de Tracy Chapman.

Las estructuras de control de bucle nos servirán para realizar las iteraciones ó acciones repetitivas, la calculadora dispone de estructuras variadas en este aspecto que nos ayudarán para que nuestro trabajo sea mucho más fácil. Estas estructuras se encuentran disponibles en el menú BRCH.

START … NEXT, START … STEP

Esta estructura nos servirá para realizar un número determinado de iteraciones fijas sin ser controladas por ninguna variable, muy útil cuando sepas cuantas iteraciones vas a realizar y no requieras de la variable de control.

Sintaxis:

```
«
  @ secuencias del programa
  Inicio Fin
  START
   @ secuencias del programa
  NEXT
  @ secuencias del programa
»
```

La estructura requiere de un número inicial y final para comenzar a iterar, **START** inicia la iteración, **NEXT** realiza la comprobación y continúa ó finaliza la iteración, el contador para las iteraciones se incrementa en 1. La estructura **START**... **STEP** es similar, la única diferencia es que antes de la palabra **STEP** debemos colocar el valor con el que deseamos incrementar el contador, en el

caso de la estructura anterior es 1, pero muchas veces queremos incrementos de 2, 3, etc, ó también decrementos, en este caso de -1, -2, etc.

Ejemplo:

```
«
  → obj n
  «
    1 n
    START
      obj
    NEXT
  »
»
```

Podríamos llamar al ejemplo 'COPIAR', lo que hace es tomar un objeto cualquiera de la pila y el número de copias a realizar, inicia el contador en 1 y continuará hasta el número de copias, por cada iteración va devolviendo a la pila una copia del objeto. Veamos un ejemplo con la otra estructura.

Ejemplo:

```
«
  0 → cad n
  «
    cad SIZE 'n' STO
    ""
    n 1
     START
       "*" +
      -1
     STEP
     cad XOR
  »
»
```

Lo que realiza el programa es codificar una cadena cualquiera. Y aquí introducimos otro concepto interesante, no siempre vamos a trabajar con los argumentos que recoge un programa sino que necesitaremos de otras variables para usarlos internamente y lo que hacemos en el ejemplo es inicializar 'n' en 0, esta es la manera de crear variables que el programa usará internamente. Además estamos utilizando la estructura **START** ... **STEP**, como habrán notado antes de la instrucción **STEP** hay un número -1, ya que la iteración será desde n hasta 1, lo que hacemos es disminuir el contador. Sería más práctico usar la estructura **START** … **NEXT**, pero es válido para el ejemplo.

FOR … NEXT, FOR … STEP

Estas estructuras son idénticas a las anteriores, la diferencia que estas poseen un contador. En realidad son las estructuras más utilizadas ya que poseen la mayoría de características para poder programar.

Sintaxis:

```
«
  @ secuencias del programa
  Inicio Fin
  FOR Contador
   @ secuencias del programa
  NEXT
  @ secuencias del programa
»
```

La estructura requiere de un número inicial y final para comenzar a iterar, **FOR** inicia la iteración, se declara una variable contador y **NEXT** realiza la comprobación y continua ó finaliza la iteración, por defecto el contador para las iteraciones se incrementa en 1, si deseas decrementos u otros incrementos debes de utilizar **STEP**, en cuyo caso debemos colocar los incrementos antes de esta instrucción.

Ejemplo:

```
«
  → n
  «
    {} 1 n
    FOR i
      i +
    NEXT
  »
»
```

El ejemplo toma un valor n, devolverá una lista con valores desde 1 hasta n. Aquí podemos notar algo bastante interesante, que es la posibilidad de adicionar elementos a una lista con solo sumarle el elemento. Existen una serie de comandos de la calculadora que funcionan de esta manera. Pero debes tener en cuenta que con las cadenas y otras listas esta misma operación tiene diferentes resultados.

Ejemplo:

```
«
  0 → lista n
  «
    lista SIZE 'n' STO {}
    n 1
    FOR i
      lista i GET +
      -1
    STEP
  »
»
```

En este ejemplo hacemos lo que el comando **REVLIST**. Obsérvese que estoy utilizando para los nombres de variables locales las minúsculas, ésta puede ser una buena opción a la hora de programar ya que te permite hacer un distingo entre las variables que utilizas y los comandos.

DO ... UNTIL ... END

Esta es otra estructura iterativa, la diferencia con las anteriores es que requiere de una condición para finalizar la iteración.

Sintaxis:

```
«
  @ secuencias del programa
  DO
    @ secuencias del programa
  UNTIL @ Condición
  END
  @ secuencias del programa
»
```

La estructura se inicia con **DO**, luego vienen las secuencias a ejecutar. La palabra **UNTIL** determina el fin de las secuencias y **END** se encarga de realizar la comprobación que determinará el fin ó repetición del bucle. Las condiciones son las mismas que para las estructuras **IF**. Debemos aclarar también que la iteración siempre se lleva a cabo por lo menos una vez, que es un poco el propósito de esta estructura.

Ejemplo:

```
«
  10 RAND * IP 0 → r n
```

```
  «
    DO
      "ADIVINA EL NUMERO MAGICO"
      "" INPUT OBJ→ 'n' STO
      CASE n r >
      THEN
        "MAS BAJO...!!!"
      END n r <
      THEN
        "MAS ALTO...!!!"
      END
        "ACERTASTE!"
      END
      MSGBOX
    UNTIL r n ==
    END
  »
»
```

Muy bien, ya vamos avanzando, aquí está un ejemplo mucho más complejo. En primer lugar creamos una variable aleatoria 'r' e inicializamos 'n' en cero; iniciamos el bucle y pedimos al usuario que adivine el número y lo almacenamos en 'n', comprobamos si 'n' es mayor que 'r' y devolvemos el mensaje correspondiente. Luego comprobamos si 'n' es menor que 'r' y devolvemos el mensaje correspondiente. Finalmente, si no se cumple ninguna de las dos condiciones anteriores, devolvemos el mensaje si acertó. Para mostrar cualquiera de los mensajes utilizamos el comando **MSGBOX**. Todo el proceso anterior se repetirá hasta que 'n' sea igual a 'r'.

Podemos realizar el mismo ejemplo de la siguiente manera:

```
«
  10 RAND * IP 0 → r n
  «
    DO
      "ADIVINA EL NUMERO MAGICO"
      ""
      INPUT
      OBJ→ 'n' STO
      n r >
      "MAS BAJO...!!!"
      n r <
      "MAS ALTO...!!!"
      "ACERTASTE!"
      IFTE
      IFTE
      MSGBOX
    UNTIL r n ==
    END
```

```
    »
»
```

El uso de **IFT** ó **IFTE** frente a las estructuras **IF** es aconsejable cuando vas a evaluar pequeñas secuencias como por ejemplo elegir entre dos ecuaciones ya que te vas a ahorrar un montón de bytes aunque el programa se te hace poco legible. Te recomiendo que ejecutes el programa paso a paso, así podrás ver cual es el proceso que sigue.

Ejemplo:

Usando IF … THEN … ELSE … END	Usando IFTE

`« → a b`
`  «  IF 'a>b'`
`  THEN '(a-b)/2'`
`  ELSE '(b-a)/2'`
`  END →NUM`
`  »`
`»`
	`« → a b`
	`  « a b >`
	`  '(a-b)/2'`
	`  '(b-a)/2'`
	`  IFTE`
	`  »`
	`»`

En el primer caso comprobamos si 'a' es mayor que 'b' y devolvemos una ecuación para el caso verdadero y otra para el caso contrario y la evaluamos. Aquí también una recomendación importante: vean que estoy ejecutando →**NUM** al final de la estructura y no después de cada ecuación, con este tipo de acciones me estoy ahorrando velocidad y bytes, debido a que en cualquier caso me devolverá una ecuación, no veo la necesidad de evaluar las ecuaciones de forma separada. En el segundo caso el comando **IFTE** requiere de tres argumentos: la condición, la operación si la condición es verdadera y la operación si la condición es falsa, fíjense que no es necesario evaluar los objetos algebraicos, **IFTE** se encargará de eso. El comando **IFTE** es muy útil sobre todo si lo quieres incluir dentro de una ecuación, lo que me lleva al siguiente ejemplo donde podrás observar todo lo que se puede hacer con la calculadora y que hace lo mismo que los ejemplos anteriores:

```
«
  → a b 'IFTE(a>b,(a-b)/2,(b-a)/2)'
»
```

WHILE … REPEAT … END

Esta es otra estructura iterativa, la diferencia con la anterior es que requiere de una condición para finalizar la iteración y no se ejecuta nunca si la condición de prueba es falsa.

Sintaxis:

```
«
  @ secuencias del programa
  WHILE @ Condición
  REPEAT
    @ secuencias del programa
  END
  @ secuencias del programa
»
```

La estructura se inicia con **WHILE**, La palabra **REPEAT** que determina si se ejecutará el bucle. La palabra **END** determina el fin de las secuencias a ejecutar. Esta estructura es muy útil si no quieres que el bucle se realice ni siquiera una vez si no se cumple la condición.

Ejemplo:

```
«
  0 → a b c
  «
    WHILE a b MOD
    REPEAT
      a b MOD 'c' STO
      b 'a' STO
      c 'b' STO
    END
    b
  »
»
```

El programa calcula el MCD (Máximo Común Divisor) de 'a' y 'b'.

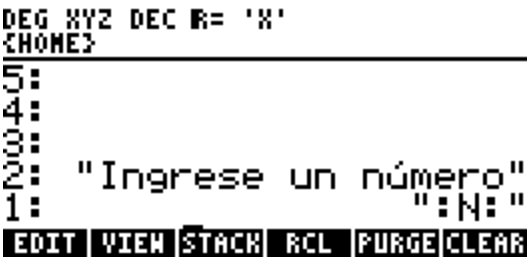
Ingreso de Datos

"Too Few Arguments" en la pantalla de mi calculadora, uhm, me pregunto ¿Qué habré olvidado?.

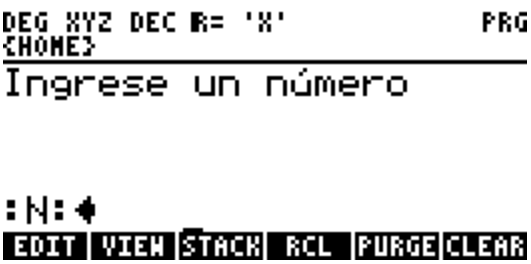
Un programa requiere de un orden en el ingreso de datos. La manera común es utilizar la pila para realizar este proceso, pero la calculadora dispone de comandos especiales que cumplir este objetivo, además permite darle una buena presentación a tus programas.

Usando INPUT

Uno de los comandos para obtener datos es **INPUT**, que nos presenta una pantalla con un título y un valor por defecto en un editor similar a la línea de comandos. Para realizar esto, el comando necesita de dos argumentos: una cadena en el nivel 2 que será el título de la pantalla y otra cadena en el nivel 1 que será texto que aparezca en la línea de comandos y que puede ser una etiqueta.



Al ejecutar el comando **INPUT** obtendríamos:



Luego, si se acepta la entrada, es decir, se presiona **[ENTER]**, **INPUT** retorna los datos introducidos como una cadena y que generalmente se convertirán a objetos conocidos con el comando →**OBJ**. Si se presiona **[CANCEL]**, la primera vez borra la línea de comandos y la siguiente cancela la ejecución del programa.

Además existe una excelente alternativa que convierte a **INPUT** en una buena opción para entrada de datos, el argumento del nivel 1 puede tener el siguiente formato:

{ "cadena" { fila columna } modo(s) }

Donde:

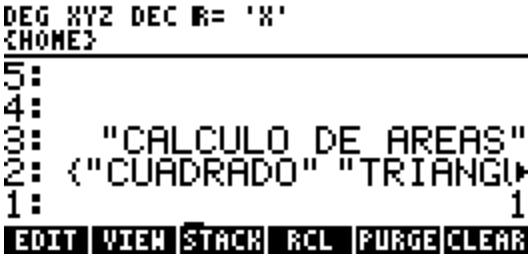
cadena: será el texto inicial, que aparezca en la línea de comandos
fila y *columna*: serán la ubicación del cursor en la línea de comandos.
modo(s): será el modo por defecto con el que se presentará la línea de comandos.

Para los modos puedes utilizar:

α, para introducir una cadena de caracteres directamente.
ALG, para introducir objetos algebraicos.
V, para realizar una comprobación de la línea de comandos.

Usando CHOOSE

CHOOSE se usa para elegir entre varias opciones. Requiere de 3 argumentos: en el nivel 3 de la pila una cadena que será el título de la lista, en el nivel 2 una lista con los elementos entre los cuales elegir y en el nivel 1 el número de elemento por defecto a seleccionar, generalmente se coloca 1.



Al ejecutar el comando **CHOOSE** tendríamos lo siguiente:

Luego de su ejecución **CHOOSE** retorna a la pila dos valores si se presiona OK ó [**ENTER**], en el nivel 1 de la pila devuelve 1 (*Verdadero*) y en el nivel 2 devuelve el objeto seleccionado. Y si se presiona CANCL ó [**CANCEL**] retorna el valor 0 (*Falso*) indicando que se canceló la operación. Con estos valores puedes tomar la decisión adecuada para continuar con el programa.

Al igual que **INPUT**, **CHOOSE** también tiene opciones adicionales para el segundo argumento. Se pueden ingresar dentro de la lista como elementos, otras listas conteniendo dos elementos, en este caso lo que muestra **CHOOSE** es el primer objeto y si se acepta alguna opción se devuelve el segundo objeto.

Usando INFORM

Y la mejor de las opciones para entrada de datos debe ser **INFORM**, que permite describir un pedido mediante título, texto descriptivo y de ayuda, además de un tipo de objeto específico.

Al ejecutar el comando **INFORM** tendríamos lo siguiente:

El primer argumento es una cadena de caracteres que representará el título, luego una lista de etiquetas para cada campo, cuyos valores se desea obtener, como tercer argumento un valor de configuración del cuadro de diálogo que puede ser un número que indicará el número de columnas, una lista con el número de columnas ó una lista con el número de columnas y el ancho de los campos, el cuarto argumento es una lista con valores predeterminados para cada campo y finalmente como último argumento una lista con valores de reconfiguración, que aparecerán cuando se presione la opción de menú Reset. Luego de su ejecución **INFORM** devuelve en la pila dos valores, si se ha presionado el menú OK ó la tecla [**ENTER**], el primer valor será un 1 (*verdadero*) y el segundo valor devuelto será una lista con los datos ingresados en el orden en que fueron pedidos. Si se presiona CANCL ó [**CANCEL**] el comando devolverá un 0 (*falso*), indicando que se canceló la operación.

Además como en el caso de **CHOOSE** e **INPUT**, el segundo argumento de **INFORM** (Nivel 4) también posee opciones adicionales y pueden ser de la forma:

{ {"Etiqueta" "Ayuda" Tipo1 Tipo2 Tipo3 … TipoN } … }

Donde:

Etiqueta: viene a ser el texto descriptivo del campo.
Ayuda: viene a ser el texto de ayuda del campo.
Tipo1 hasta *TipoN*: viene a ser el tipo de objeto(s) que el campo puede aceptar.

Ejemplo:

```
«
  "Area de un triángulo:
(Ingrese los lados)"
  { ":a:
:b:
```

```
:c:" { -1 4 } V }
  INPUT
  OBJ→ 0
  → a b c p
  «
    a b c + + 2 / 'p' STO
    p p a - * p b - * p c - * √
    "AREA" →TAG
  »
»
```

En este ejemplo se muestra el uso de **INPUT**, se trata de un programa que calcula el área de un triángulo, para lo cual necesitará el valor de los tres lados.

Otro ejemplo:

```
«
  "CALCULO DE AREAS:"
  { {"Cuadrado" 1}
  {"Triángulo" 2}
  {"Círculo" 3} }
  1
  CHOOSE
  IF 1 ==
  THEN
  → fig
  «
    CASE fig 1 ==
    THEN
      "AREA DE UN CUADRADO"
      { { "Lado:" "Ingrese el lado del cuadrado" 0 } }
      1
      { }
      { }
      INFORM
      IF 1 ==
      THEN
        EVAL 2 ^ "AREA" →TAG
      END
    END fig 2 ==
    THEN
      "AREA DE UN TRIANGULO"
      { { "Base:" "Ingrese la base del triángulo" 0 }
        { "Altura:" "Ingrese la altura del triángulo" 0 } }
      1
      { }
      { }
      INFORM
      IF 1 ==
      THEN
```

```
      EVAL * 2 / "AREA" →TAG
    END
  END
    "AREA DE UN CIRCULO"
    { { "Radio:" "Ingrese el radio del círculo" 0 } }
    1
    { }
    { }
    INFORM
    IF 1 ==
    THEN
      EVAL 2 ^ π * "AREA" →TAG
    END
  END
»
END
»
```

Este es un ejemplo mucho más completo, se trata de un programa capaz de calcular el área de cualquier figura que se selecciona de una lista. Aquí se podrá notar claramente el uso tanto de **CHOOSE** como de **INFORM**.

Presentación de Datos

Luego de la ejecución de un programa ó durante ésta, debemos de presentar algunos resultados. Al igual que en el caso anterior la calculadora posee excelentes comandos para presentar datos. Vayamos directamente a los ejemplos.

Usando MSGBOX

MSGBOX muestra un cuadro de mensaje y el único argumento que necesita es una cadena. Pero si queremos adicionarle números u otros objetos, simplemente tenemos que convertirlos en cadenas, sumarlos y listo.

Ejemplo:

```
«
  0 → r a
  «
    π r 2 ^ 'a' STO
    εEL AREA DEL CIRCULO ES:ɳ
    a →STR +
    MSGBOX
  »
»
```

Usando DISP

DISP muestra texto en el número de fila indicado de la pantalla (el número de fila pueden ser de 1 hasta 7). Para realizar este proceso necesita 2 argumentos, primero una cadena y luego el número de fila.

Ejemplo:

```
«
  → nom
  «
    nom 1 DISP
```

```
    0 FREZZE
  »
»
```

En el ejemplo se aprecia otro comando que se denomina **FREEZE**, este comando se encarga de congelar la pantalla en un área determinada y que se le proporciona como argumento, estos pueden ser:

Argumento	ZONA CONGELADA
0	Toda la pantalla
1	El área de estado
2	La pila
3	El área de estado y la pila
4	Los menús
5	Los menús y el área de estado
6	La pila y los menús
7	Toda la pantalla

Finalmente para mejorar la presentación conviene borrar la pantalla, para ello utilizaremos el comando **CLLCD**. Este comando se encarga de borrar la zona ocupada por la pila y la línea de estado, que es la zona que utiliza **DISP**.

Ejemplo:

```
«
  "Ingrese una cadena:"
  { "" α }
  INPUT 1
  → c i
  «
    c c SIZE 21
    START
      " " +
    NEXT
    DUP + 'c' STO
    CLLCD
    DO
      c i i 21 + SUB
      4 DISP .5 WAIT
      IF i 22 ==
      THEN
        1 'i' STO
      ELSE
        1 'i' STO+
      END
      KEY
    UNTIL 0 ≠
```

```
        END
        DROP
    »
»
```

El propósito de éste ejemplo mostrar en medio de la pantalla una cadena de caracteres rotando hasta que se presiona cualquier tecla. La cadena debe ser introducida como dato al inicio del programa.

Trabajando con Ecuaciones

El trabajo en la calculadora con las ecuaciones tiene desventajas sólo en cuanto a velocidad. Pero surge la necesidad de usarlos y no hay otra forma de hacerlo, así que manos a la obra. Es hora de otro café, ¿ya notaron que me encanta?

La calculadora dispone de muchas maneras para crear, evaluar ecuaciones ó algebraicos, pero no vamos a hablar mucho de eso, si no de algunos otros detalles que nos servirán para realizar nuestros programas.

Ejemplo:

```
«
  → x
  «
    'F(X)=EXP(X)-2' DEFINE
    { } 1 x
    FOR i
      i F +
    NEXT
    'F' PURGE
  »
»
```

El programa consiste en un evaluador de una función para ciertos valores, en este caso desde el número 1 hasta 'x'. La idea del ejemplo es demostrar como se crean las funciones mediante un programa y eliminarla cuando éste termina; muy útil en aplicaciones de métodos numéricos ya que tienes la posibilidad de que el usuario ingrese la ecuación y crear la función con diferentes ecuaciones. En este caso se devolverá una lista con los valores de la función para cada argumento. Si analizas esto en la pila verás que la función F, no es más que un programa.

Aclaro que no es la única forma, ya que podrías evaluar las expresiones algebraicas con el comando "**|**", pero el proceso se haría más lento ya que tendrías que recuperar las ecuaciones por cada iteración.

Ejemplo:

```
«
  → x
  «
    { } 1 x
    FOR i
      '2^X-X^2' { X i } | +
    NEXT
  »
»
```

En este caso se recomienda realizar las operaciones de pila:

Ejemplo:

```
«
  → x
  «
    { } 1 x
    FOR i
      2 i ^ X 2 ^ - +
    NEXT
  »
»
```

IMPORTANTE:

- En general se recomienda las operaciones de pila, puedes utilizar por ejemplo **OVER**, **DROP**, **ROT**, **DUP** que son funciones rápidas (por lo menos para UserRPL) y tus programas serán mucho más pequeños. Nuevamente, ésta es solo una recomendación, sobretodo para aquellos a los que les interesa la optimización.

- Si se trata de sumatorias utilicen las funciones de la calculadora, no utilicen **FOR** ... **NEXT** ó algo por el estilo y verán que les resulta más provechoso.

- Los programas pueden utilizarse como funciones propias de la calculadora y dentro de las ecuaciones ó expresiones algebraicas, lo que deben de hacer estos es devolver algún valor ó valores para efectuar las operaciones.

- Vuelvo a recalcar que algunos comandos de la calculadora, evalúan directamente las expresiones algebraicas, como por ejemplo las estructuras **IF** ... **THEN** ... **END**, **IFT**, **IFTE**, así que no es necesario evaluarlas en las condiciones.

- Pueden usarse expresiones para obtener datos de matrices sin que estas sean evaluadas. Veremos con más detalle este aspecto en el capítulo correspondiente a Matrices.

Trabajando con Matrices

Para muchos usuarios el trabajo con las matrices representa un dolor de cabeza, pero si empleas las técnicas adecuadas verás que resulta sencillo trabajar con ellas. Los comandos para trabajar con matrices se encuentran disponibles en el menú MATR.

Ejemplo:

```
«
  0 0 → mat m n
  «
    mat SIZE EVAL { m n } STO
    1 m
    FOR i
      1 n
      FOR j
        mat { i j } GET
      NEXT
    NEXT
  »
»
```

En este sencillo ejemplo lo que hacemos es devolver todos los valores de la matriz a la pila. Los pasos son sencillos: obtenemos la matriz, calculamos el tamaño con el comando **SIZE**. En este punto debemos saber que este comando es muy funcional ya que puede calcular el tamaño de listas, matrices, cadenas, gráficos, etc. Para nuestro caso devolverá en una lista el número de filas y el número de columnas, las cuales almacenamos en las respectivas variables. Nótese también la manera de almacenar estas variables, es una manera muy práctica de almacenar varias variables. Luego vamos a un bucle para las filas 'i' y el bucle interno para las columnas 'j', luego con el comando **GET** obtenemos el número de la fila columna de la matriz { i j }.

El comando **GET** también trabaja con expresiones algebraicas, de hecho i j son expresiones algebraicas. Podemos hacer entonces, algo como esto:

Ejemplo:

```
«
  @ secuencias del programa
  mat { 'i+1' 'j-1' } GET
  @ secuencias del programa
»
```

Para introducir valores en la matriz haremos uso del comando **PUT**.

Ejemplo:

```
«
  0 0 → mat m n
  «
    mat SIZE EVAL { m n } STO
    1 m
    FOR i
      1 n
      FOR j
        IF 'i == j'
        THEN
          mat { i j } 1 PUT 'mat' STO
        END
      NEXT
    NEXT
    mat
  »
»
```

En este ejemplo reemplazamos la diagonal de una matriz por unos. Pero el programa pudo haber sido optimizado de la siguiente forma:

```
«
  0 0 → mat m n
  «
    mat SIZE EVAL { m n } STO
    1 m
    FOR i
      1 n
      FOR j
        IF 'i == j'
        THEN
          'mat' { i j } 1 PUT
        END
      NEXT
    NEXT
    mat
  »
»
```

El comando **PUT** dispone de la capacidad de trabajar sobre variables que contienen matrices y en este caso ya no hay necesidad de escribir nuevamente la matriz. Además que la operación se llevará a cabo de una manera mucho más rápida.

Existen muchas funciones para trabajar con matrices como por ejemplo **REPL**, **SUB**, que te permiten reemplazar y extraer respectivamente partes de una matriz y que son de gran ayuda a la hora de programar.

También una matriz puede ser usada dentro de expresiones algebraicas por ejemplo en operaciones de sumatorias en donde los datos fila y columna serían los argumentos de la variable que contiene la matriz, algo así como una especie de función:

Ejemplo: `'mat(2,5)'`

Los valores de los argumentos en este caso pueden ser reemplazados por variables, además pueden ser usados con el comando **STO**, que trabaja directamente sobre la matriz y realiza las veces del comando **PUT** y con **EVAL** que realizaría las veces del comando **GET**.

Trabajando con Listas

Las listas si que son *"listas"*, vienen a ser un arma (con el perdón de los pacifistas) muy poderosa a la hora de programar, una de las ventajas de usar listas es que éstas pueden contener otros objetos, al igual que con las matrices puedes insertar, eliminar, cambiar elementos y una opción muy importante es que puedes ejecutar un programa sobre elementos de la lista. Los comandos para trabajar con listas se encuentran disponibles en el menú LIST.

Al aplicar un comando sobre listas muchos de ellos se ejecutan sobre la lista y otras actúan sobre cada elemento de la lista. Puede resultar entonces muy práctico, realizar operaciones sobre toda la lista y no sobre cada elemento:

Ejemplo:

Ejecutando en la línea de comandos:

```
{ 4 2 } SQ
```

obtendremos

```
{ 16 4 }
```

y funciona igual con muchos otros comandos.

Otro ejemplo:

Ejecutando

```
{ 1 2 } { 3 4 } +
```

obtendremos

```
{ 1 2 3 4 }
```

Como se puede observar se ha añadido la lista 2 a la lista 1, en el caso de que queramos sumar los elementos tendremos que utilizar el comando especial **ADD** y no la operación '**+**'.

Además la calculadora dispone de comandos relacionados sólo con operaciones de listas. De acuerdo con esto, podríamos hacer por ejemplo una extensión del comando **MAX**:

Ejemplo:

```
«
  SORT REVLIST HEAD
»
```

El programa requerirá una lista en el nivel 1 devolverá el número máximo de la lista.
Supongamos ahora que queremos convertir cada uno de los elementos de una lista en cadenas tendríamos que realizar un bucle, pero existe una forma más práctica:

Ejemplo:

```
«
  1
  «
    →STR
  »
  DOSUBS
»
```

Nótese que el subprograma no recoge ninguna variable y por lo mismo no se ejecutará, sino se devolverá a la pila. El comando **DOSUBS** necesita como argumentos el número de elementos sobre los que se ejecutará el programa que es el primer argumento de **DOSUBS**. Con este tipo de comandos puedes crear algoritmos mucho más complejos y que te permitan realizar tus programas de manera más eficaz.

Trabajando con Gráficos

Los gráficos son muy importantes a la hora de programar y la calculadora nos brinda la oportunidad de realizar programas interesantes.

Una de las utilidades es la gráfica de funciones ó datos. Mediante la programación y el manejo de las variables del sistema como **PPAR** podemos graficar funciones del tipo y escala que nosotros queramos. La calculadora nos brinda un gran número de comandos para realizar estas funciones y obtener buenos resultados.

Ejemplo

```
«
  RAD
  { PICT PPAR } PURGE
  POLAR θ INDEP
  '3*COS(2*θ)' STEQ
  DRAW
  { # 0d # 0d } PVIEW
  0 FREEZE
»
```

Lo que hacemos en el ejemplo es definir primero el sistema angular, borramos las variables **PICT** y **PPAR**. **PICT** es una variable del sistema, es el lienzo sobre el cual se dibuja. Luego definimos el tipo de gráfico, la variable y la ecuación, a continuación **DRAW** realiza el gráfico sobre **PICT**. En este punto podemos visualizar la imagen mediante **PICTURE** o realizar el procedimiento que se ve en el ejemplo. **PVIEW** realiza la misma función que el comando **DISP**, pero con gráficos.

También podemos crear presentaciones que no tengan que ver con gráfica de funciones y para esto sólo debemos trabajar sobre la variable del sistema **PICT** ó sobre gráficos creados por nosotros mismos. Estos comandos están disponibles en los grupos GROB y PICT de menú [**PRG**].

Para trabajar con gráficos es necesario tener una idea sobre las coordenadas de pantalla sea cual sea el modo en el que trabajemos. Las coordenadas se pueden representar de dos maneras:

```
(5,8)
{ # 11 # 30 }
```

Ejemplo:

```
«
  { # 11 # 30 }
  { # 60 # 100 }
  LINE
  { # 0d # 0d } PVIEW
  0 FREEZE
»
```

La mejor manera es el segundo modo, para ello deberás tener en cuenta que la coordenada superior izquierda viene a ser (0,0) y la coordenada inferior derecha será (130,63).

Las operaciones de este tipo se realizan sobre **PICT** y para visualizarlas debes utilizar **PVIEW** ó detener las operaciones con el teclado, lo que vimos en un capítulo anterior. Las operaciones como **REPL** y **SUB** también son aplicables a los gráficos y puedes hacer cosas interesantes con ellos. También en el grupo GROB puedes encontrar comandos como **GXOR** que te permitirán crear efectos en los gráficos. Puedes crear gráficos de textos y en general de cualquier objeto, recuperar la pantalla actual, etc.

Veamos un ejemplo mucho más completo:

```
«
  1 # 15d # 15d BLANK
  5 2 50 32
  → d g m n x y
  «
    ERASE { # 0d # 0d } PVIEW
    { # 4d # 1d }
    { # 125d # 62d }
    BOX
    DO
      .1 WAIT
      PICT
      x R→B y R→B 2 →LIST
      GROB 15 15 0E308FF0CFF1EFF3EFF3FFF7FFF7FFF7FFF7BFF76FF3AEF
      349F18FF00E30
      REPL
      PICT
      m R→B n R→B 2 →LIST
      g REPL
      x 'm' STO
      y 'n' STO
```

```
{ x y }
CASE d 1 ==
  THEN
    CASE x 5 == y 2 == AND
      THEN
        15 4
      END y 2 ==
      THEN
        { -15 15 } 3
      END x 5 ==
      THEN
        { 15 -15 } 2
      END
        -15 1
    END
  END d 2 ==
  THEN
    CASE x 110 == y 2 == AND
      THEN
        { -15 15 } 3
      END y 2 ==
      THEN
        15 4
      END x 110 ==
      THEN
        -15 1
      END
        { 15 -15 } 2
    END
  END d 3 ==
  THEN
    CASE x 5 == y 47 == AND
      THEN
        { 15 -15 } 2
      END y 47 ==
      THEN
        -15 1
      END x 5 ==
      THEN
        15 4
      END
        { -15 15 } 3
    END
  END
  CASE x 110 == y 47 == AND
    THEN
      -15 1
    END y 47 ==
    THEN
      { 15 -15 } 2
    END x 110 ==
    THEN
      { -15 15 } 3
```

```
          END
            15 4
        END
      END
      'd' STO STO+ KEY
    UNTIL 1 ==
    END DROP
  »
»
```

En el ejemplo se muestra el uso de algunas funciones gráficas, lo que hace el programa es hacer rebotar una pelota por toda la pantalla hasta que se presiona una tecla. Todos estos comandos para trabajar con gráficos están explicados uno a uno en el manual de usuario.

Depuración de Programas

Estoy pensando que escribir, mientras le doy los últimos sorbos a mi última taza de café, antes de terminar este manual…

Al terminar de codificar un programa, éste no siempre te funciona (recuerden: errar es humano, pero procuren no ser humanos siempre). Hallar el error representará una tarea tediosa. La calculadora me permite depurar los programas basado en las operaciones de pila, donde podrás apreciar en detalle el proceso que sigue tu programa y te sea sencillo saber la causa del fallo.

Para depurar un programa debes de ir al grupo RUN del menú [**PRG**] ingresa el nombre del programa a depurar en el nivel 1 de la pila y ejecuta DBUG lo cual inicializará el programa. Para ir por el programa paso a paso ejecuta SST; si el programa necesita argumentos estos deben ser introducidos en la pila antes de ejecutar este comando. Hay otra función análoga para ejecutar el programa paso a paso, que es SST↓, la diferencia con el anterior es que éste comando también depurará los subprogramas que se encuentran dentro del programa principal.

Para detener la depuración en cualquier momento puedes utilizar KILL.

Pero no necesariamente puedes realizar de esta manera la depuración. Puedes detener la ejecución de tu programa colocando la instrucción **HALT** en la posición que consideres dentro de tu programa. Al ejecutarse tu programa y llegar a ésta instrucción éste se detendrá y podrás hacer el seguimiento como en el caso anterior.

Ejemplo:

```
«
  @ secuencias del programa
  HALT
  @ secuencias del programa
»
```

Apéndice: Un calendario

(Seguramente algunos van a necesitar una buena taza de café)

```
@ Filename: CALENDARIO
@ Author: Roger G. Broncano
@ Description:
@   Programa para mostrar un calendario personalizado
@ Revision date: 28/04/2001
@ Stack diagram:
@     ( --> )

«
  0 DATE IP DATE FP 100 * DUP IP SWAP FP 10000 * 4 DUPN 0
  → g M D A x m d a y
  «
    ERASE
    23 64
    FOR i
      #9 i R→B 2 →LIST #120 i R→B 2 →LIST
      LINE
      10
    STEP
    9 121
    FOR i
      i R→B #14 2 →LIST #64 i R→B 2 →LIST
      LINE
      16
    STEP
    PICT { #14 #8 } "DO  LU  MA MIE JUE VIE SA" 1 →GROB REPL
    PICT { #9 #7 } PICT { #9 #7 } { #121 #13 } SUB NEG REPL
    PICT { #74 #1 } "[+][-]MES-RGB" 1 →GROB REPL
    PICT RCL 'g' STO
    ERASE
    { #0 #0 } PVIEW
    1
    DO
      IF 1 ==
      THEN
        PICT { #0 #0 } g REPL
        A 1000000 / 1 100 / + M +
        { "SUN" "MON" "TUE" "WED" "THU" "FRI" "SAT" }
        TIME TSTR 1 3 SUB POS
        1 - 16 * 12 + 'x' STO
```

```
  15 'y' STO
  PICT { #9 #1 }
  { "Enero" "Febrero" "Marzo" "Abril" "Mayo" "Junio"
    "Julio" "Agosto" "Setiembre" "Octubre" "Noviembre"
    "Diciembre" }
  M GET " - " + A →STR TAIL TAIL + 1 →GROB REPL
  1 { 31 28 31 30 31 30 31 31 30 31 30 31 } M GET
  A 4 MOD 0 == M 2 == AND 1 0 IFTE +
  FOR i
    PICT x R→B y R→B 2 →LIST i 2
    →GROB REPL
    IF 'd==I AND M==m AND A==a OR M==4 AND i==10 OR
        M==6 AND i==26 OR M==9 AND i==25 OR M==10 AND i==2 OR
        M==11 AND i==15'
    THEN
      PICT x 2 - R→B y 1 - R→B 2 →LIST
      #15 #9 BLANK NEG GXOR
    END
    IF x 108 ==
    THEN
      IF y 55 ==
      THEN
        15 'y' STO
      ELSE
        10 'y' STO+
      END
      12 'x' STO
    ELSE
      16 'x' STO+
    END
  NEXT
END
IFERR 0 WAIT
THEN
  DROP 91
END
IP 'x' STO
CASE x 95 ==
  THEN
    1
    IF M 12 ==
    THEN
      1 'M' STO 'A'
    ELSE
      'M'
    END
    INCR
  END x 85 ==
  THEN
    1
    IF M 1 ==
    THEN
      12 'M' STO 'A'
```

```
          ELSE
            'M'
          END
          DECR
        END
        0
      END
    UNTIL 0 ==
    END
  »
»
```

Palabras Finales…

Creo que se terminó, hemos llegado al final de este manual, y ahora que me percato, también se terminó mi taza de café. Espero que haya resultado provechoso leer este manual y no se olviden de seguir expandiendo sus conocimientos, en verdad que ya no me queda más que decir, por lo menos por ahora…

Roger G. Broncano
rogerbroncano@hotmail.com
http://www.gaak.com/hpuser