

1. INTRODUCCION.

La siguiente documentación es una referencia para la programación en las calculadoras serie 48GX y 49G. De ninguna manera este texto se constituye como parte oficial o autorizado por los fabricantes, su objetivo es orientar al usuario (de la manera más explícita posible), y entender los diferentes comandos que presenta la calculadora. Este texto va dirigido a estudiantes de las diferentes ramas de la ingeniería y a cualquier persona interesada en la programación en calculadoras HP.

Existen muchos lenguajes de programación en las calculadoras Hewlett Packard, entre los que se encuentran algunos lenguajes, entre ellos, USER-RPL, SYS-RPL, ASSEMBLER, Y CODIFICADO, y es necesario realizar un estudio más profundo, y disponer de tiempo para poder depurar y entender los mencionados lenguajes, debido a la dificultad de entendimiento y por la necesidad de realizar estudios adicionales de Informática, principalmente para los dos últimos lenguajes citados.

El lenguaje de programación que generalmente es utilizado por el usuario, es el lenguaje USER-RPL, y será estudiado en el presente texto, como parte fundamental, también se realizará un estudio sobre el lenguaje SYS-RPL, cuya explicación, se encuentra más adelante, y finalmente, programación gráfica, que si bien no es un lenguaje independiente, requiere también, aplicación y se estudiara, en este caso, de manera casi independiente; la programación gráfica, al igual que el lenguaje SYS-RPL, será explicada durante el desarrollo del presente texto.

2. TIPOS DE VARIABLES.

Dentro la programación en la calculadora, se manejan cuatro tipos de variables, las primeras son las variables propias del sistema, y las otras tres, muy funcionales, son las variables globales, las variables locales, y las variables compiladas.

Las variables del sistema, que son propias del sistema, es decir, los comandos que utiliza la calculadora para realizar las diferentes operaciones o ejecutar diferentes trabajos. Las siguientes variables no pueden tener el mismo nombre que las variables del sistema.

Las variables globales, son aquellas que el usuario puede crear asignando un nombre a cualquier objeto, y puede editarse, evaluarse, manipularse o borrarse. Este tipo de variable se almacena en la memoria disponible de la calculadora, donde es posible acceder a estas de forma rápida.

Las variables locales, son las que se utilizan dentro de un solo programa, y son temporales, es decir, solo existen durante la ejecución del programa, no puede ser compartido con otros

subprogramas, a menos que vuelvan a ser definidas, puede ser editado dentro la propia ejecución del programa, y ocupan la memoria interna de la calculadora. Dependiendo la necesidad, una variable local puede convertirse en una variable global. Las variables locales tienen la ventaja de hacer que los programas se ejecuten más rápidamente que al utilizar variables globales.

Las variables compiladas, tienen las mismas características de las variables locales, con la diferencia que estas pueden ser utilizadas en otros subprogramas, sin necesidad de definir las nuevamente. Se caracterizan por llevar una flecha ← antes del nombre de la variable.

La programación con las diferentes variables será aclarado gradualmente al realizar los siguientes programas y ejemplos; algunos de estos serán representados tal como se encuentra en la pantalla, mientras que otros mostrarán el programa en sí, esto por la dificultad de mostrar todo el contenido del programa en una sola pantalla de la calculadora.

3. PROGRAMACION.

Un programa en la calculadora, es un objeto delimitado por los símbolos << " *programa* " >>, y puede contener una secuencia de números, operaciones o comandos que se desean ejecutar de forma automática para realizar una tarea. Un objeto, es cualquier comando, número, lista, programa, etc. que se encuentra en el nivel 1: de la pila.

Al realizar un programa y ubicarlo en el nivel 1: de la pila, se debe guardar con el nombre de preferencia del usuario, delimitando este por los símbolos algebraicos '*NOMBRE*' STO, con lo que se logra almacenar en la memoria del usuario dicho programa. Se debe tomar en cuenta la disponibilidad de memoria que tiene la máquina para realizar un programa de cualquier tipo.

Existen muchas formas de realizar los programas, y generalmente se obedece a la plataforma o preferencia que tiene el usuario, respecto a la entrada o salida de datos. Durante el desarrollo, se representan unos ejemplos de programación, en distintas plataformas de un mismo programa, si no se especifica de otra manera.

Es importante considerar que la estructura de cualquier programa, tiene siempre la secuencia:

Entrada de Datos

Procesamiento de Datos

Salida de Datos

Y este será el desarrollo, tomando en cuenta, que el texto, se divide en tres capítulos fundamentales:

1. Programación USER-RPL, que es la de mayor uso y entendimiento por parte de los usuarios, además de ser la programación más independiente, en cuanto a los comandos.
2. Programación SYS-RPL, que es una interacción entre la programación USER-RPL, y una codificación interna del sistema de la calculadora, con números en formato binario. Este tipo de programación es muy conveniente, cuando se desea realizar trabajos especiales, o muy particulares, toma en cuenta los comandos SYSEVAL, y LIBEVAL. Es muy importante considerar que este tipo de programación es de mucho cuidado, es decir, el mal uso de los mencionados comandos, podrían causar la pérdida de memoria actual de la calculadora, y en casos extremos, el daño del Hardware o memoria física de la calculadora. También es importante recalcar que la programación SYS-RPL, varía entre las calculadoras 48 y 49, por lo tanto, se debe verificar una correspondencia de codificación entre las dos Calculadoras.
3. Programación Gráfica, que también es muy importante, al momento de realizar programas que requieran entornos gráficos, y su estudio es mas avanzado y de alguna manera, con mayor complejidad.

Por lo tanto, en el primer apartado (Programación USER-RPL), se encuentran separados cada componente de un programa. En primer lugar se cuenta con el apartado de ENTRADA DE DATOS, a continuación SALIDA DE DATOS, y finalmente PROCESO DE DATOS, todo esto aplicado a la programación en la calculadora. El orden de aprendizaje, es debido a que la parte más importante (Proceso de datos), se realizará de manera más independiente.

En base al primer apartado, se desarrollara el posterior contenido del texto, es decir, Programación SYS-RPL, y programación con aplicaciones o entornos Gráficos.

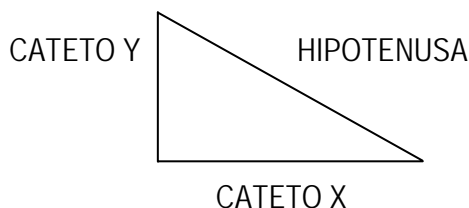
CAPITULO I: PROGRAMACIÓN USER-RPL

I. 1. INTRODUCCION DE DATOS.

El formato de introducción de datos en la programación USER-RPL está siempre en función a las preferencias del usuario; el ingreso de datos puede ser conveniente a partir del esquema que se prefiere utilizar, el tiempo que se tarda en la introducción de dichos datos, o el tiempo que tarda en activarse la plantilla o formato de introducción de datos. En primer lugar tendremos unos ejemplos mostrando la entrada y salida de datos directamente desde la calculadora, y luego utilizando las plantillas de introducción de datos, y finalmente, los formatos de salida de resultados.

Programa 1.

El siguiente programa halla la hipotenusa de un triángulo rectángulo, y se presenta en distintas formas de programa, el usuario debe decidir cual es de su preferencia o conveniencia. Cabe mencionar que las unidades de medición que se utilizan para la resolución de los problemas, en todo momento deben ser las mismas, o correspondientes.



El ingreso de datos es desde la pila, de la siguiente manera:

Cateto X: 4 cm.
Cateto Y: 3 cm.

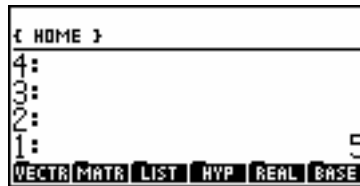
Entonces en la calculadora, los datos de entrada son:



Como los datos se encuentran en la pila, se puede realizar las siguientes operaciones en el orden indicado:

$\leftarrow x^2$ $\leftarrow \blacktriangleright$ $\leftarrow x^2$ + $\sqrt{\quad}$
 SQ SWAP SQ ADICION RAIZ CUADRADA

Dando como resultado, Hipotenusa = 5cm.



Las anteriores operaciones pueden ser compiladas o agrupadas en un programa, y se debe proceder de la siguiente manera:

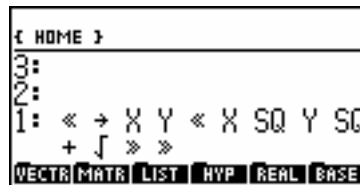
`<< SQ SWAP SQ + √ >>`

que realizará la misma operación de forma automática.



Programa 2.

En el siguiente programa, se utilizarán variables locales. El ingreso de datos es igual al anterior (desde la pila).



Se nota claramente que en el programa se llama a las variables locales, para realizar cada operación independientemente, y no se requiere ya utilizar el comando SWAP.

Programa 3.

En el siguiente programa, se utilizarán variables locales, en combinación con el modo algebraico. El ingreso de datos es desde la pila.



El modo algebraico es una herramienta de gran potencia, cuando se manejan expresiones algebraicas muy difíciles de depurar o entender. En expresiones algebraicas, generalmente se coloca el comando (NUM, esto debido a que, mediante la configuración de la calculadora, es posible que los resultados sean devueltos de forma simbólica, y no de forma numérica.

I. 1.1. USO DEL COMANDO INPUT.

Este es el comando de introducción de datos mas utilizado en la programación, debido a su facilidad y su baja complejidad, en comparación a los otros comandos del mismo tipo.

El comando INPUT requiere de dos argumentos, como se muestra a continuación;

1. "Cadena de Caracteres" En la que se especifica la variable que se va a introducir, indicar las unidades requeridas, precauciones, etc.
2. { " " α V ## ALG } Una lista de contenido variable, con parámetros especificando el tipo de dato, en función del uso que tendrá el comando INPUT, este argumento es estudiado a detalle en los ejemplos siguientes. En la representación se ve el argumento de la forma mas completa; por lo tanto se debe considerar que no todos los parámetros indicados son imprescindibles.

Programa 4.

En este programa se realizará una introducción de datos personalizada, mediante el comando INPUT en su forma más sencilla, e irá aumentando su complejidad gradualmente.

Se realizará el mismo programa anterior.

```
<< "Introduce Cateto X:" { " " } INPUT OBJ→  
"Introduce Cateto Y:" { " " } INPUT OBJ→  
→ X Y << X SQ Y SQ + √ >> >>
```

Se puede hacer una modificación en el anterior programa, de la siguiente manera:

```
<< "Introduce Cateto X  
y Cateto Y:" { " " } INPUT OBJ→  
→ X Y << X SQ Y SQ + √ >> >>
```

en el cuál se requiere un espacio entre los datos ingresados, para no confundir con una sola cifra.

Otra manera un poco más indicativa en cuanto a la introducción de datos, es de la siguiente manera:

```
<< "Introduce Cateto X  
y Cateto Y" { ":Cateto X:  
:Cateto Y:" 11 } INPUT OBJ→  
→ X Y << X SQ Y SQ + √ >> >>
```

en el cuál la operación entre llaves, indica la posición en la cuál se deben introducir los datos, con su respectiva referencia, y el numero 11, indica la posición inicial del cursor; si este número es negativo (-11), se indica a la calculadora, que el cursor estará en modo Sobre escribir (■), de lo contrario, el cursor estará en modo Insertar (◀). Cuando a este número no se lo coloca, de forma automática, la calculadora coloca el cursor en la última posición.

El comando INPUT tiene la opción de verificar si existe algún error en la introducción de datos mediante la letra V, de la siguiente manera:

```
<< "Introduce Cateto X
    y Cateto Y" {"Cateto X:
    :Cateto Y:" V -11 } INPUT OBJ→
    → X Y << X SQ Y SQ + √ >> >>
```

Si ocurre algún error en la introducción de datos, la calculadora, mediante este comando, no permite salir de la plantilla de ingreso de datos (a menos que la ejecución del programa sea cancelada por el usuario) , hasta que este sea realizado de forma correcta.

Programa 5.

Si en el programa se requiere el ingreso de datos alfabéticos o alfanuméricos, y se decide utilizar el comando INPUT, se debe proceder de la siguiente manera:

El siguiente programa pide el nombre y el teléfono de una persona, los agrupa en una lista, y la guarda en la variable DATO1:

```
<< "Nombre completo:" { " " α } INPUT
    "Telefono:" " " INPUT OBJ→
    2 →LIST 'DATO1' STO >>
```

Se ve que para el ingreso de datos alfabéticos, no es necesario utilizar el comando OBJ→ , ya que este separa los miembros de una cadena, y en este caso, no es adecuado.

Programa 6.

El siguiente programa, requiere que se coloque una expresión Algebraica en el nivel 1 de la pila de comandos, para guardarla en la variable EQ:

```
<< "Introduce la Ecuación"
    { " ' ' " α 2 ALG } INPUT OBJ→
    STEQ >>
```

En este caso es necesario incluir dentro la cadena de caracteres, las comillas particulares de objeto, con el propósito de evitar un error al ejecutar el comando OBJ→, esto debido a que la calculadora no puede reconocer un objeto algebraico, como una cadena de caracteres.

Es posible realizar muchas combinaciones con el comando INPUT, variando el contenido de las llaves. Finalmente, se puede realizar el siguiente programa 7, tomando en cuenta todos los parámetros que son soportados por el comando INPUT, considerando que es este caso no se requiere de una ecuación algebraica.

Programa 7.

El siguiente programa requiere el nombre de una persona, su apellido, y su edad, verifica algún error en la introducción de datos, agrupa estos datos en una lista y la guarda en la variable DATO2:

```
<< "Datos:"  
  {":Nombre:  
    :Apellido:  
    :Edad:" α V -9 } INPUT OBJ→  
  3 →LIST 'DATO2' STO >>
```

Según el caso, será necesario desactivar o activar el modo alfabético manualmente, es decir con la tecla α. A continuación se presentan los parámetros soportados por el comando INPUT:

{ " <i>Dato Introducido</i> "	α	V	##	ALG }
NECESARIO	OPCIONAL	OPCIONAL	OPCIONAL	OPCIONAL

El empleo de estos parámetros o la preferencia de algunos de ellos, así como el uso de las llaves, es decisión del usuario, en el texto se especifica de la manera más general. También Se debe considerar que el empleo de estos parámetros deben tener correspondencia, para evitar errores o una mala interpretación en el trabajo de la calculadora, al momento de ejecutar el programa.

I. 1.2. USO DEL COMANDO INFORM.

Este comando presenta otra plantilla de introducción de datos estructurada y más compleja; en principio se estudiará en su forma más sencilla, ya que posteriormente se combinará con operadores lógicos, para poder completar la capacidad que nos presenta esta plantilla.

Este comando tiene la siguiente estructuración:

1. "Título" Posición en la que va el título o nombre del programa que se ejecuta.

2. { { "Variable 1" "Especificación 1" ## }
 { "Variable 2" "Especificación 2" ## }
 { "Variable 3" "Especificación 3" ## }

 { "Variable n" "Especificación n" ## } }

3. { ## } Número de columnas que se desean generar o mostrar durante la ejecución. Es importante recalcar que, cuando en este campo no se especifica ningún valor, automáticamente la calculadora asume el valor de una columna, por lo tanto, solamente podrán visualizarse y editarse 4 variables; en el caso de que el valor especificado sea 2, podrán visualizarse 8 variables, y así sucesivamente.

4. { "Valor 1" "Valor 2" "Valor n" } Valores secundarios o auxiliares de las n variables consideradas.

5. { "Valor 1" "Valor 2" "Valor n" } Valores por defecto predeterminados antes de la ejecución, de las n variables consideradas.

6. INFORM

Detalle.

1. Este campo tiene la finalidad de dar una referencia de la plantilla, es decir, se debe colocar entre comillas, el nombre del programa, u operación que se realizará durante la ejecución.
2. Consta de una serie de sub-listas, cada una conteniendo 3 posiciones; la primera posición indica cual es la variable correspondiente a ser llenada o editada; la segunda posición contiene un detalle o referencia de la variable que se edita respectivamente, y la tercera posición contiene un número, que indica el tipo de variable que se está manejando. La asignación de estos números es como sigue:

Asignación	Tipo de Variable
0	Número Real
1	Número Complejo
2	Cadena de Caracteres
3	Matriz de Números Reales
4	Matriz Compleja
5	Lista
6	Nombre de Variable

3. Se coloca el número de columnas que se desea sean visibles en la plantilla. Cuando se utilizan demasiadas variables, se debe tener cuidado de lograr que todas entren en la plantilla, ya que las variables que no pueden ser incluidas, automáticamente son excluidas de la introducción de datos.
4. En esta lista, se colocan los valores auxiliares asignados a cada variable, y no son visibles en primera instancia, estos se logran visualizar ejecutando el comando RESET, incluido en el menú de la propia plantilla INFORM. Esta lista generalmente se la coloca vacía, salvo casos muy especiales y que realmente sean necesarios valores auxiliares.



5. Al igual que la anterior lista, en esta se colocan los valores asignados a cada variable, pero por defecto, es decir que al ingresar a la plantilla, esta ya se presentará con los valores asignados, y se tiene la opción

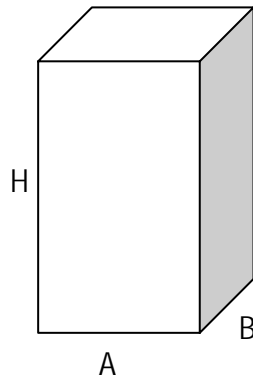
de verificar los valores o cambiarlos, si es necesario. Esta lista también puede ser dejada vacía, pero su uso es más frecuente que la anterior.

6. El comando INFORM, que ejecuta la plantilla preparada para la introducción de datos. En la plantilla INFORM se tiene la opción de aceptar o cancelar la introducción de datos, devolviendo a la pila el valor de 1, si la introducción de datos fue aceptada, o el valor de 0, si la introducción de datos fue cancelada, para luego ser verificada la operación mediante operadores lógicos, y continuar con la ejecución del programa. Como se mencionó antes, este paso será excluido mientras no se realice el estudio de los operadores lógicos.



Programa 8.

El siguiente programa utiliza la plantilla INFORM, y realiza el cálculo del volumen ocupado por un cuerpo prismático ortogonal de base A x B, y altura H.



```
<< "CALCULO EN PRISMA"  
  { "A:" "Valor A de la Base" 0 }  
  { "B:" "Valor B de la Base" 0 }  
  { "H:" "Valor de la altura" 0 } }  
  { 2 }  
  { }  
  { } INFORM DROP  
  EVAL → A B H  
<< A B * H * >> >>
```

Programa 9.

El siguiente programa utiliza la plantilla INFORM, y realiza el cálculo del volumen y la superficie ocupada por un cuerpo prismático ortogonal de base A x B, y altura H; y guarda el valor del volumen en la variable VOL, y el valor de la superficie en la variable SUP.

```
<< "CALCULO EN PRISMA"
  {"A:" "Valor A de la Base" 0 }
  {"B:" "Valor B de la Base" 0 }
  {"H:" "Valor de la altura" 0 } }
  { 2 }
  { }
  { } INFORM DROP
  EVAL → A B H
  << A B * H * 'VOL' STO
    A B * 2 * H A * 2 *
    H B * 2 * + + 'SUP' STO >> >>
```

Programa 10.

El siguiente programa utiliza la plantilla INFORM, para realizar una lista con el nombre, apellido, edad y teléfono de una persona, y guarda estos datos en la variable DATO3. Se debe tomar en cuenta que , la plantilla INFORM reconoce los nombres o caracteres alfabéticos, como cadenas de caracteres.

```
<< "DATOS PERSONALES"
  {"NOM:" "NOMBRE DE LA PERSONA" 2 }
  {"AP:" "APELLIDO DE LA PERSONA" 2 }
  {"EDAD:" "EDAD ACTUAL" 0 }
  {"TEL:" "TELEFONO" 0 } }
  { 1 }
  { }
  { } INFORM DROP
  'DATO3' STO >>
```

En este caso no se tuvo que evaluar la operación (EVAL), ya que la plantilla INFORM, da como respuesta, una lista de los valores introducidos, entonces, no es necesario separar esta lista para luego volver a unirla. Realizando una pequeña modificación en el anterior programa, combinando con la estructura de secuencia lógica IF - THEN - END, se completa la operación que realiza el comando INFORM. La estructura de secuencia lógica tiene el siguiente formato:

- IF:** Realiza la pregunta si la última operación es *Verdadera/Falsa* con los valores 1/0 respectivamente. Como se dijo anteriormente, la salida que tiene el comando INFORM, es dependiendo si se terminó el llenado de datos, pulsando posteriormente la tecla OK, en el nivel 1 de la pila, estará el número 1, indicando que la operación fue completada, entonces es verdadera; de lo contrario, si se canceló el ingreso de datos, en el nivel 1 de la pila, la plantilla devolverá al nivel 1 de la pila, el valor de 0, es decir, operación falsa.
- THEN:** El comando "*Entonces*", que al verificar si la respuesta fue verdadera, el programa realizará el procesamiento o desarrollo del programa, de lo contrario, se irá hasta la posición END.
- END:** Es la parte de la estructura en la cuál se indica que la operación lógica a concluido, y se puede seguir con el desarrollo del programa, por lo tanto, no es necesario o condicionante, que el comando END, sea colocado al final del programa.

Completando la funcionalidad del comando INFORM, se completa el anterior programa:

```
<< "DATOS PERSONALES"  
  {"NOM:" "NOMBRE DE LA PERSONA" 2 }  
  {"AP:" "APELLIDO DE LA PERSONA" 2 }  
  {"EDAD:" "EDAD ACTUAL" 0 }  
  {"TEL:" "TELEFONO" 0 } }  
  { 1 } { } { } IF INFORM THEN  
  'DATO3' STO END >>
```

Con la estructura IF - THEN - END, se realizó la verificación que realiza la plantilla INFORM, en cuanto al ingreso de datos.

I. 2. SALIDA DE DATOS.

Como ya hemos visto, la salida de datos puede representarse en el nivel 1: de la pila (dependiendo de cuantas fueran las respuestas), o también puede ser almacenado en variables para su posterior uso. A continuación se presentará al usuario algunas formas de representar la salida de datos, tomando en cuenta que, al igual que con el ingreso de datos, la forma de representar las salidas está sujeta a las preferencias o conveniencias del usuario.

I. 2.1. USO DEL COMANDO →TAG.

Programa 11.

El siguiente programa se guarda en la variable ESF.ENT, y halla el volumen de una esfera de radio "r", guarda el valor del radio en la variable global RA, luego llama al subprograma ESF.SAL, y este devuelve el volumen de la esfera con una etiqueta indicando que la respuesta es el volumen (→TAG).

Programa ESF.ENT

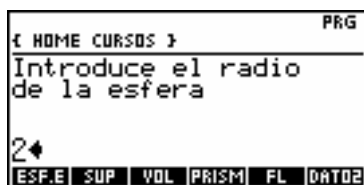
```
<< "Introduce el radio  
de la esfera"  
{ " " } INPUT OBJ → 'RA' STO  
ESF.SAL >>
```

Subprograma ESF.SAL

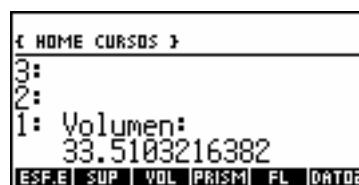
```
<< '(4/3)*π*RA^3' →NUM  
"Volumen" →TAG >>
```

En el anterior caso, fue necesario definir primero el programa de salida, ya que al realizar en primer lugar el programa ESF.ENT, y ejecutarlo, no se completa la operación. El usuario debe realizar las respectivas pruebas para aclarar el orden de programación.

Al realizar el cálculo del volumen en una esfera de radio = 2, se tiene:



Entrada (ESF.ENT)



Salida (ESF.SAL)

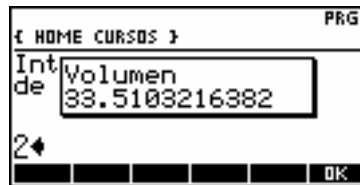
I. 2.2. USO DEL COMANDO MSGBOX.

Programa 12.

La salida que realizará el siguiente programa utilizará la ventana MSGBOX, mostrando en esta, el resultado del ejemplo anterior. El programa ESF.ENT ya se encuentra definido.

```
<< '(4/3)*π*RA^3' →NUM  
"Volumen:  
" SWAP + MSGBOX >>
```

Esfera de radio = 2:



En el anterior programa, se hizo un artificio antes del comando SWAP, al realizar un ↵ (tecla de punto), esto para lograr que la cadena de caracteres sea más explícita, en lo que refiere a su representación en la ventana MSGBOX. El usuario notará las diferencias con diferentes pruebas que realice.

I. 2.3. USO DEL COMANDO DISP.

Programa 13.

El formato de salida de datos del siguiente programa, utilizará el comando DISP, combinado con el comando WAIT para mostrar la respuesta durante 2 segundos.



El comando DISP, muestra el contenido de cualquier cadena de caracteres, por fracciones de segundo, por lo que, mediante el comando WAIT, se determina su duración, en este caso, 2 segundos.



La pantalla de la calculadora tiene en total 7 niveles para poder representar cualquier cadena mediante el comando DISP, siendo así, el numero 1, la posición más alta en la pantalla, y 7, la posición más baja en la pantalla de la calculadora.

I. 3. PROCESO DE DATOS.

En este apartado, se realizará un estudio de los diferentes ciclos o Programación estructurada que posee la calculadora; esta posee tras grandes grupos de estructuras, las estructuras condicionales, las de repetición, y las de detección de errores

I. 3.1. ESTRUCTURAS CONDICIONALES.

Las estructuras condicionales o de prueba, son de gran importancia para la realización de programas de forma más eficiente y para realizar una menor generación de errores. La calculadora posee las siguientes estructuras condicionales:

Estructura IF... THEN... END, estructura condicional con una sola salida o respuesta, la verdadera. Si la respuesta a la condicional es falsa, el trabajo no es efectuado, y la calculadora termina el trabajo de la estructura, es decir, se llega al comando END.

Estructura IF... THEN... ELSE... END, que al igual que la anterior, verifica si alguna operación es posible de realizar, mediante una condicionante, además se tiene la opción de realizar un trabajo adicional, como respuesta negativa.

Estructura CASE... END, que toma a la estructura IF... THEN... END las veces que sean necesarios, de manera que la respuesta no sea única, es decir que se pueden tomar muchos casos como respuestas posibles. También se puede utilizar la estructura IF... THEN... ELSE... END, pero en este caso, se estaría aumentando la complejidad de la estructura CASE...END, al realizar operación sobre operación.

I. 3.1.1. USO DE LA ESTRUCTURA IF... THEN... END.

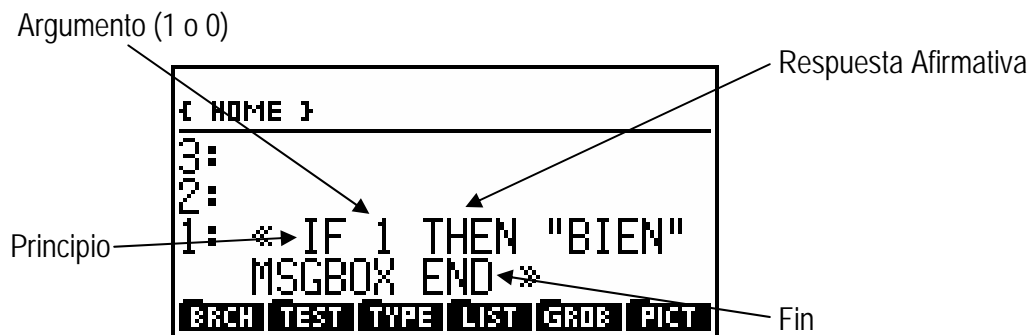
Como se vio rápidamente en un anterior ejemplo, esta estructura tiene la función de verificar si una operación es verdadera o falsa, mediante los valores de 1, como verdadero, o 0, como valor falso, y en función a esta condicionante realizar un determinado trabajo.

IF: Requiere un argumento 1 o 0, es el comando que verifica si una operación es verdadera o falsa, respectivamente; se debe tomar en cuenta que la calculadora toma como valor falso, cualquier numero entero que sea distinto a 1. Trabaja de la siguiente manera: Mediante operadores lógicos, se determina la veracidad de cualquier trabajo que fue efectuado anticipadamente, si la respuesta es afirmativa, se ejecuta la subrutina comandada por THEN. Si la respuesta fuera falsa, el ciclo o rutina termina sin opción a realizar ninguna otra operación auxiliar.

THEN: En este punto, se realiza el trabajo que fue designado para una respuesta verdadera, se debe tomar en cuenta, que dentro de cualquier subrutina, se pueden generar otros programas o sub-subrutinas.

END: Es el comando que indica la finalización de la estructura, en el caso que el proceso no se haya efectuado, es decir, la respuesta fue falsa, de igual manera se llegará a este punto, para continuar con la ejecución del programa si este fuera el caso.

Esquema de la estructura IF... THEN... END:



Programa 14.

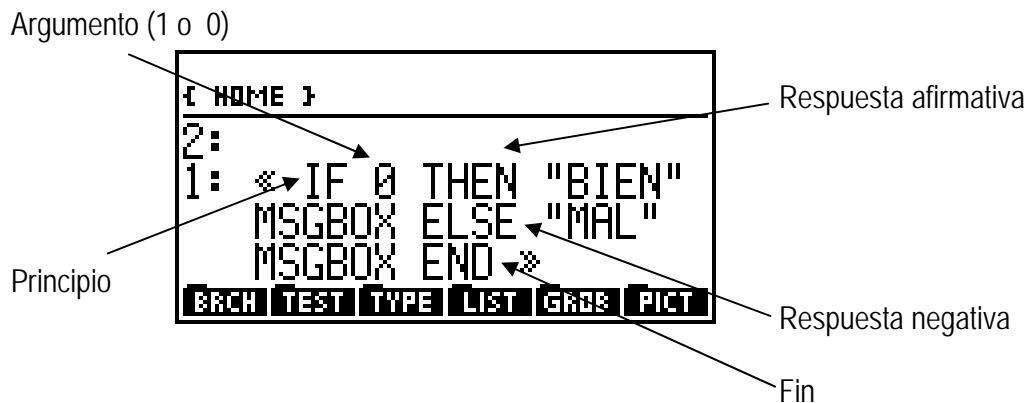
El siguiente programa, pide un código secreto predeterminado como "XY22", si es correcto, muestra la plantilla "Acceso permitido", de lo contrario termina.

```
<< "Introduce la clave" { " " } INPUT
  IF "XY22" ==
  THEN "Acceso permitido" 1 DISP 2 WAIT
  END >>
```

I. 3.1.2. USO DE LA ESTRUCTURA IF... THEN...ELSE... END.

Esta estructura trabaja de la misma manera que la estructura IF... THEN...END, con la diferencia que se tiene la opción de realizar una tarea adicional o auxiliar, mediante el comando ELSE.

ELSE: En el caso de que en la verificación IF, se haya determinado falsedad, la maquina omitirá la subrutina gobernada por THEN, y se dirigirá a la subrutina ELSE, en esta se podrán realizar trabajos adicionales o auxiliares, dependiendo el caso o necesidad del programa.



Programa 15.

El siguiente programa pide la edad, y verifica si es mayor o menor de edad.

```
<< "Edad" { " " } INPUT
  OBJ→ IF 18 ≥
  THEN "Mayor de edad" MSGBOX
  ELSE "Menor de edad" MSGBOX
  END >>
```

I. 3.1.3. USO DE LA ESTRUCTURA CASE... END.

La función de esta estructura, es determinar una secuencia de posibles respuestas, no necesariamente una sola, por ejemplo, si se desea saber si la edad de una persona esta entre 0 a 10 años, entre 10 a 20 años, entre 20 a 30 años, etc., mediante una sola estructura.

CASE: Es el comando que indica el inicio de la ejecución de la estructura CASE... END, después van las diferentes condicionales que son requeridas para la verificación, siempre cada condicional es como una sub-estructura, por lo tanto, se debe tener cuidado, con el comando END principalmente, es decir:

```
<<... CASE
      IF... THEN... END
      IF... THEN... END
      IF... THEN... END
      END >>
```

Luego de realizar todas las verificaciones, hay la posibilidad de colocar una cláusula por defecto, es decir, sin importar que las condicionales anteriores fueron verdaderas o falsas. Esta cláusula no es indispensable:

```
<<... CASE
      IF... THEN... END
      IF... THEN... END
      IF... THEN... END
      Cláusula por defecto
      END >>
```

END: Indica el final de la estructura.

Programa 16.

El siguiente programa verifica si un número es positivo, negativo o 0.

```
<< "Numero?" { " " } INPUT OBJ→ → X
<< CASE
      IF X 0 > THEN "positivo" END
      IF X 0 < THEN "negativo" END
      IF X 0 == THEN "es cero" END "Numero " SWAP
      + MSGBOX END >> >>
```

Como se ve, en la estructura CASE, resulta de mucha facilidad utilizar las variables locales, para facilitar el programa, y su entendimiento.

I. 3.2. ESTRUCTURAS DE REPETICION.

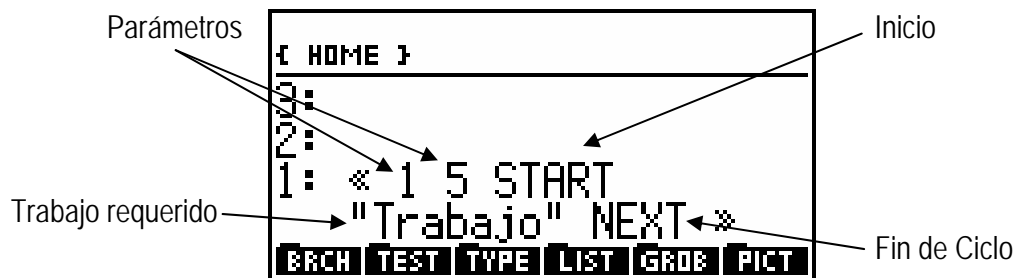
Estas estructuras, tienen diferentes funciones, y depende del usuario el campo de aplicación en el que sea requerido.

I. 3.2.1. USO DE LA ESTRUCTURA START... NEXT.

Esta estructura ejecuta un proceso repetitivo de una determinada rutina o trabajo, y siempre se ejecuta por lo menos, una vez; es muy útil cuando se desea, dependiendo el caso, realizar el mismo trabajo un determinado numero de veces.

START: Requiere como parámetros, dos números en la pila, en el nivel dos se coloca el límite inferior de la generación, y en el nivel 1, se coloca el límite superior de la generación. Los límites deben ir de inferior a superior, de lo contrario, se realiza el ciclo una sola vez. El incremento es de una unidad, es decir números enteros.

NEXT: Este comando, indica que se completó un ciclo, y que se continuará (en el caso de no haber terminado), realizando el mismo ciclo hasta llegar al límite superior de la generación, incrementando la generación de uno en uno.



Programa 17.

Realizar un programa que pida durante 3 veces consecutivas, el nombre y el teléfono de tres personas, los agrupe en una lista independiente, luego forme una lista global, y la guarda en la variable DATO4:

```
<< 1 3 START
"Nombre completo:" { " " α } INPUT
"Telefono:" { " " } INPUT OBJ→
2 →LIST NEXT
3 →LIST 'DATO4' STO >>
```

Programa 18.

El siguiente programa pide cuantas veces se desea realizar el cálculo de la hipotenusa de un triángulo rectángulo, y devuelve los resultados en una caja MSGBOX.

```
<< "Numero de cálculos?" {" "} INPUT OBJ→  
1 SWAP START  
"Introduce Cateto X:" {" "} INPUT OBJ→  
"Introduce Cateto Y:" {" "} INPUT OBJ→  
→ X Y << X SQ Y SQ + √ "Hipotenusa:  
" SWAP + MSGBOX >>  
NEXT >>
```

I. 3.2.2. USO DE LA ESTRUCTURA START... STEP.

Esta estructura tiene la misma función que la anterior, con la diferencia que se pueden generar incrementos fraccionales, y generar decrementos. Su uso es muy particular, y generalmente el empleo de esta estructura es perfectamente reemplazada, por su utilidad, por la estructura FOR...STEP, que se estudia en el apartado 8.4.

START: Requiere como argumentos, dos números en la pila, en el nivel dos se coloca el límite inferior de la generación, y en el nivel 1, se coloca el límite superior de la generación. Los límites no necesariamente tienen que ser en orden ascendente, y se determina el incremento o decremento con el comando STEP.

STEP: Este comando requiere un argumento (número real o entero), e indica que se completó un ciclo, y que se continuará (en el caso de no haber terminado), realizando el mismo ciclo hasta llegar al límite de la generación, incrementando o decrementando la generación en función al paso (argumento) que le fue designado.

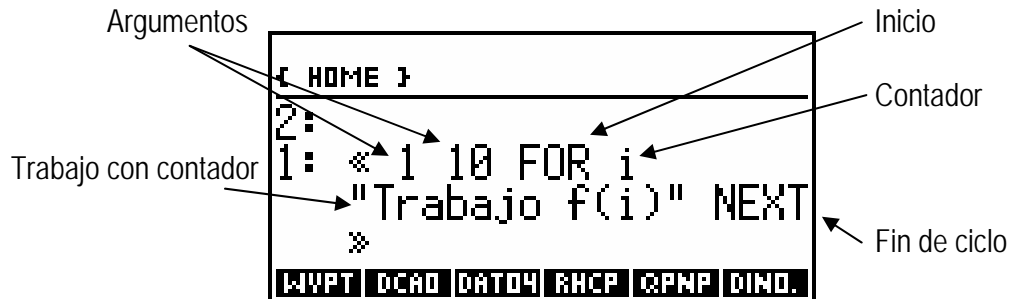
I. 3.2.3. USO DE LA ESTRUCTURA FOR... NEXT.

Esta estructura es muy potencial, se podría decir que es la más utilizada en la programación de la calculadora, por la aplicación en los diversos programas o rutinas requeridos por el usuario.

FOR: Requiere dos argumentos, en el nivel 2 de la pila, el límite inferior de la generación, y el límite superior en el nivel 1 de la pila, además se asigna con una variable convencional, que puede definirse como variable local y su función es efectuar el conteo; el incremento es de uno en uno, como números enteros.

NEXT: Es el paso que requiere este ciclo, no precisa de ningún argumento, e indica que se ha completado un ciclo, y se continuará ejecutando el mismo trabajo hasta llegar al límite superior de la generación, incrementando el paso de uno en uno.

Esquema de la estructura FOR... NEXT:



Programa 19.

El siguiente programa, realiza una lista de los 10 primeros números naturales, los guarda dentro la variable 'DAT05', y saca la sumatoria de todos estos números, luego muestra la raíz cúbica de cada uno de estos números contenidos en la lista.

```
<< 1 10 FOR i
    i NEXT 10 →LIST
    'DAT05' STO
    DAT05 ΣLIST
    "Sumatoria:
    " SWAP + MSGBOX
    1 10 FOR i
    DAT05 i GET
    3 XROOT "Raiz Cubica de "
    i + ":
    " + SWAP + MSGBOX
    NEXT >>
```

Programa 20.

El siguiente programa pide la edad de un número de personas, las guarda en una lista dentro la variable local N, y determina cuantas son mayores de edad.

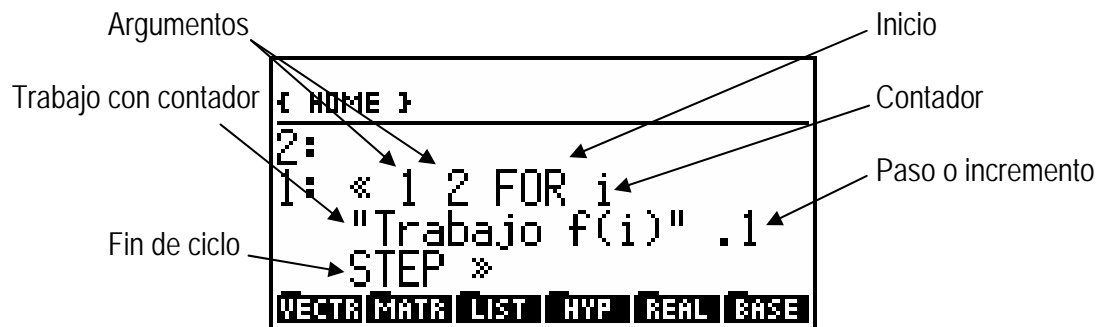
```
<< { } 0 → N NN
    << "Numero de personas?"
    { " " } INPUT OBJ→
    → NP << 1 NP FOR i
    "Edad de la persona " i + { " " } INPUT OBJ→
    N + 'N' STO NEXT
    N SIZE 1 SWAP FOR j
    N j GET IF 18 ≥ THEN NN 1 + 'NN' STO END
    NEXT
    "Mayores de edad:
    " NN + MSGBOX >> >> >>
```

I. 3.2.4. USO DE LA ESTRUCTURA FOR... STEP.

FOR: Requiere dos argumentos, en el nivel 2 de la pila, el límite inferior de la generación, y el límite superior en el nivel 1 de la pila, estos límites no tienen que ser necesariamente en orden ascendente, y pueden ser números reales o enteros, además se asigna con una variable convencional (variable local), la cuál trabaja como contador; el incremento está en función al incremento o decremento asignado como argumento antes del comando STEP.

STEP: Este comando requiere un argumento (número real o entero), e indica que se completó un ciclo, y que se continuará (en el caso de no haber terminado), realizando el mismo ciclo hasta llegar al límite de la generación, incrementando o decrementando la generación en función al paso (argumento) que le fue designado.

Esquema de la estructura FOR... STEP:



Programa 21.

El siguiente programa halla el valor de la ordenada de la recta $Y=1+1.5X$, en función a su coordenada, es decir $Y = f(X)$, entre los valores de 1 y 2, con un paso de 0.05, y halla el área debajo la recta entre esos límites. (Tomar en cuenta que mientras más pequeño el paso, mayor exactitud en los resultados).

```
<< 0 → cc
<< 1 2 FOR i
1 1.5 i * +
DUP "Para X=" i + "
f(X)=" + SWAP +
MSGBOX 0.05 * cc +
'cc' STO 0.05 STEP
"Area =" cc 4 RND +
MSGBOX >>
```

I. 3.2.5. USO DE LA ESTRUCTURA DO... UNTIL... END.

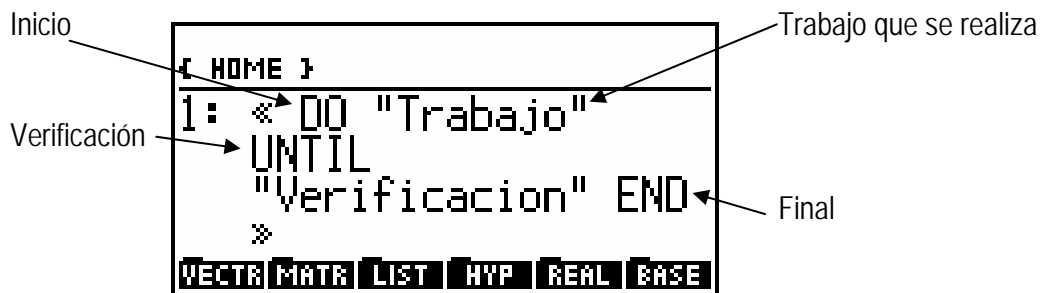
Esta es otra herramienta potencial que posee la calculadora, y su estructura es de la siguiente manera:

DO: (Hacer), que no requiere ningún argumento, es el comando que indica a la calculadora, realizar alguna operación de forma repetitiva e indefinida, mientras no se cumpla la condicionante UNTIL (Mientras).

UNTIL: (Mientras), es el comando que condiciona a la calculadora hasta cuando debe efectuar un determinado trabajo. Generalmente esta estructura trabaja combinada con una variable de cualquier tipo, por ejemplo, antes de ejecutar el ciclo, se definió la variable local A, con el valor de 1, y durante la ejecución del programa se realiza el incremento de A, hasta 10, entonces, se realiza el ciclo hasta (UNTIL), que la variable A, toma el valor de 10. Este caso será aclarado mediante ejemplo.

END: El comando que indica la finalización de la estructura, es importante saber que END, no necesariamente significa la finalización del programa, por lo tanto, puede ir acomodado en cualquier parte del programa. Internamente la calculadora realiza la estructuración del programa, y verifica si hay algún error en el programa, en este caso, el usuario debe tener la precaución de haber utilizado correcta y completamente las diferentes estructuras utilizadas en la programación.

Esquema de la estructura DO... UNTIL... END:



Programa 22.

El siguiente programa, halla el promedio de las notas de una materia (1er. Parcial, 2do. Parcial y examen final), las veces que el usuario lo requiera.

```
<< DO
  "Introduce las notas" { ":1er. Parcial:
:2do. Parcial:
:Examen final:" 15 V } INPUT OBJ→
+ + 3 / "Promedio:
" SWAP DUP 3 ROLLD + SWAP
  IF 51 ≥ THEN "
    Aprobado" +
  ELSE "
    Reprobado" +
  END MSGBOX
UNTIL
"Desea continuar? S/N" { " " α } INPUT
"N" ==
END >>
```

I. 3.2.6. USO DE LA ESTRUCTURA WHILE... REPEAT... END.

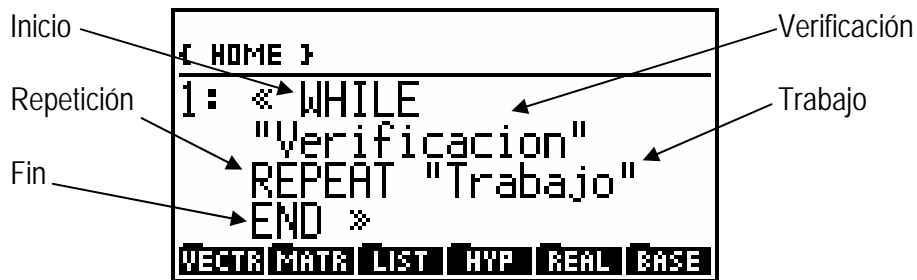
A diferencia de la estructura DO... UNTIL... END, esta en primer lugar verifica si la siguiente rutina es posible de realizar, mediante una condicionante definida antes por el usuario, entonces se deduce que al utilizar la estructura DO... UNTIL... END, la calculadora efectúa el ciclo por lo menos una vez, mientras que con la estructura WHILE... REPEAT... END, no necesariamente se tendrá que efectuar la rutina.

WHILE: (Mientras), es el comando que reconoce que la operación será factible, por ejemplo, se definió la variable local B, con el valor de 1, y se desea efectuar una rutina 5 veces, entonces hasta que el valor de B no haya sido incrementado en el proceso (REPEAT), se efectuará la misma rutina. Si el valor de B ya fue definido con el valor de 5, entonces no se realizará ningún ciclo, y la calculadora emitirá cualquier proceso hasta llegar el comando END correspondiente al ciclo que se está utilizando.

REPEAT: Es el comando que indica cual es la acción repetitiva que se deberá realizar.

END: Indica el final de la estructura.

Esquema de la estructura WHILE... REPEAT... END:



Programa 23.

El siguiente programa realiza un el cálculo de la Sumatoria, media, y desviación estándar de un ensayo estadístico en "n" muestras, siendo este número determinado por el usuario.

$$Sumatoria = \sum_1^n Muestra_i \qquad \text{Pr omedio} = \frac{\sum_1^n Muestra_i}{n}$$

$$Desvío = \sqrt{\left(\frac{\sum_1^n (Muestra_i^2)}{n} - \left(\frac{\sum_1^n Muestra_i}{n} \right)^2 \right) \times \frac{n}{n-1}}$$

```

<< "Introduce número
de muestras:" {" "} INPUT OBJ→
{ } → NN LIS
<< WHILE LIS SIZE NN <
  REPEAT "Introduce dato:" {" "} INPUT OBJ→
    LIS + 'LIS' STO
  END "Sumatoria:
  " LIS ΣLIST + MSGBOX
  "Promedio:
  " LIS ΣLIST NN / + MSGBOX
  "Desvío estándar:
  " LIS 2 ^ ΣLIST LIS SIZE / LIS ΣLIST LIS SIZE /
  2 ^ - LIS SIZE DUP 1 - / *√ + MSGBOX
>> >>

```

I. 3.3. ESTRUCTURAS DE DETECCIÓN DE ERRORES.

Estas estructuras tienen la finalidad de detectar si existe algún error mientras el programa se este Ejecutando, y de esta manera, evitar la interrupción del programa. Estas estructuras tienen un formato muy parecido a las estructuras condicionales:

Estructura IFERR... THEN... END, estructura de detección de error con una sola salida o respuesta, si se verifica error, el trabajo de la calculadora es ejecutar las operaciones que suceden al comando THEN, en el caso de no haber error, la calculadora termina el trabajo de la estructura, es decir, se llega al comando END.

Estructura IFERR... THEN... ELSE... END, que al igual que la anterior, verifica si alguna operación es objeto de error, entonces se ejecutan las operaciones comandadas por THEN, si no existe error, se tiene la opción de realizar un trabajo adicional, como respuesta negativa (ELSE).

I. 3.3.1. USO DE LA ESTRUCTURA IFERR... THEN... END.

IFERR: Verifica si existe algún error en las operaciones que continúan a partir de este comando, por ejemplo, si se realiza el producto entre una cadena y un número cualquiera, en un programa sin detector de errores, la operación es suspendida, el programa es abortado, al utilizar esta estructura, se puede subsanar este error, y realizar una nueva introducción de datos (si fuese el caso), o continuar el programa sin tomar en cuenta el error. De ninguna manera esta estructura permite que un error sea omitido, lo que se obtiene es una operación adicional para subsanar algún error de ejecución.

THEN: Comando por el cual se realizan las operaciones pertinentes antes de continuar con la ejecución del programa. En el caso de no haber error, la maquina continuará la ejecución a partir del comando END.

END: Indica el final de la Estructura.

Esquema de la estructura IFERR... THEN... END:



Programa 24.

El siguiente programa (XPOS) pide el valor de la posición X de una lista de 10 números llamada LISTA10, si hay error, se tiene que volver a ingresar dicha posición, de lo contrario, termina mostrando el valor de dicha posición.

```
<< "Ingrese Posición:" { " " } INPUT OBJ→
→ P << IFERR LISTA10 P GET
      THEN 2 DROPN
      "Posición no valida" MSGBOX
      XPOS END
>> >>
```

I. 3.3.2. USO DE LA ESTRUCTURA IFERR... THEN... ELSE... END.

De la misma manera que la anterior estructura, pero con la opción auxiliar de continuar la ejecución mediante la operación ELSE.

ELSE: Es el camino adicional en caso de que no se detecte algún error en el programa. Este comando es muy particular, es decir, su uso no es muy frecuente, por lo que su estudio es reducido a casos muy especiales, o se podría decir que es una combinación con la ejecución normal del programa.



Programa 25.

El siguiente programa, realiza la división entre dos valores ingresados desde la pila, en caso de existir error, los invierte, si el error continua, entonces los valores son multiplicados. En caso de continuar el error, significa que ambos valores no son los correspondientes a una operación algebraica normal, por lo que el programa es abortado.

```
<< → A B
<< IFERR A B / →NUM
  THEN IFERR B A / →NUM
    THEN A B * IFERR →NUM
      THEN "No se pudo
            completar el
            programa"
            MSGBOX
            6 DROPN
      ELSE "" A + "x" + B + "=
            " + A B * +
            MSGBOX
      END
    ELSE B A / "" B + "/" + A +
          "=
          " + SWAP + MSGBOX
          3 DROPN
    END
  ELSE A B / →NUM "" A + "/" + B +
        "=
        " + SWAP + MSGBOX
        DROP
END >> >>
```

Las diferentes aplicaciones, u operaciones que podrían generarse con estas estructuras, dependen de la habilidad que posea el programador, o el enfoque que se da a alguna estructura en particular; para conseguir esto, solo es necesario practicar con diferentes problemas, y ver las diferentes alternativas por las cuales se podrían optar.

CAPITULO II: PROGRAMACIÓN SYS-RPL

II.1. OBJETIVO.

El objetivo de la programación SYS-RPL, es el de realizar operaciones o trabajos, de una manera que parecería imposible realizar, utilizando el lenguaje de usuario estándar. El lenguaje SYS-RPL, como se dijo al principio del texto, es un complemento que interactúa con el lenguaje de usuario, por lo tanto, es importante tener en principio el conocimiento del lenguaje de usuario. Se debe considerar también, que el uso de estos comandos, se lo debe realizar con cuidado, ya que con un mal uso, fácilmente se puede causar pérdida total de la memoria en la calculadora, por lo tanto, es una buena previsión realizar una copia de seguridad, subiendo los programas actuales, a la computadora.

SYS-RPL, o lenguaje del sistema, consta básicamente de dos comandos, SYSEVAL, y LIBEVAL, los cuales ejecutan un determinado trabajo de acuerdo a la codificación o al argumento que se coloca en el nivel 1 de la pila; este argumento, es un número en formato entero binario; y una lista de dichas codificaciones (Argumentos) están disponibles en números de base hexadecimal, en la red Internet únicamente; ya que hasta el momento no se cuenta con un texto en nuestro medio que considere dicha codificación. El usuario puede investigar en algunas paginas de Internet, o investigar las paginas de Internet mencionadas en las referencias al final del texto.

NOTA: Es importante mencionar que estos comandos, difieren según el modelo de la calculadora que se utiliza, entonces, se debe cuidar de estar utilizando el argumento adecuado.

Los comandos explicados en principio, se los define de la siguiente manera, y características:

1. **SYSEVAL**, es el comando que sirve para realizar tareas muy especiales, y en base al argumento o dirección que se utiliza, va directamente a una localización en la memoria interna, es un comando muy de extrema potencia, por lo que se debe cuidar de estar haciendo lo correcto; requiere de los siguientes argumentos:
 1. #AAAFFFh Argumento (Codificación) o dirección interna de la memoria de la calculadora, en formato entero binario.
 2. SYSEVAL Que realiza la ejecución o trabajo de la mencionada dirección interna.

Considerar que, además de estos argumentos, en algunos casos se requieren otros argumentos adicionales, de acuerdo al trabajo que se desea realizar. Esta idea será aclarada con los ejemplos que se realizaran durante el desarrollo.

2. **LIBEVAL**, que tiene las mismas propiedades, con la diferencia que estos comandos realizan operaciones, para facilitar el entorno o uso de las calculadoras, en otras palabras, utiliza las librerías propias, menús propios de la calculadora, para una mayor optimización, en cuanto al desarrollo de programas, también se podría decir, que sirven para la ejecución del programa, pero no son parte del programa.

1. **#AAFFh** Argumento (Codificación) o dirección interna de la librería interna de la calculadora, en formato entero binario.
2. **LIBEVAL** Que realiza la ejecución o trabajo de la mencionada librería interna.

Considerar que, además de estos argumentos, en algunos casos se requieren otros argumentos adicionales, de acuerdo al trabajo que se desea realizar. Esta idea será aclarada con los ejemplos que se realizarán durante el desarrollo

Finalmente, se puede mencionar que el estudio de estos comandos, mas que todo, es una depuración de todos los comandos utilizados por las calculadoras Hewlett Packard, por ende, el presente estudio, se restringe a los comandos SYSEVAL y LIBEVAL que tiene uso mas frecuente; como comentario, se puede decir que los comandos SYSEVAL son mas de 3000; y los comandos LIBEVAL son también un numero parecido.

❖ *El Concepto de Entero Binario, (BINARY INTEGER) proviene de la convención de los programadores de la calculadora, para denominar al número en forma más específica, por lo tanto, considerar en lo futuro que al hablar de un número Entero Binario, no se trata de un número Binario necesariamente.*

II.2. USO DEL COMANDO SYSEVAL.

En primer lugar, tenemos que adoptar la siguiente simbología, para determinar si se requieren argumento adicionales:

Símbolo	Tipo	Ejemplo
\$	Cadena de caracteres	"ABCDEDEF"
id	Nombre global	'A'
array	Matriz	[[1 2] [3 4]]
grob	Objeto gráfico	graphic 131x64
#	Numero binario	# 15777h
symb	Objeto simbólico	'A^2+B^2'
list	Lista de objetos	{ A CD XZ ... }

Para interpretar la pila y el uso de los comandos, se especifica, bajo la simbología propuesta, si es necesario utilizar argumentos adicionales, antes de la flecha →, y la consecuencia o resultado, bajo la misma simbología, después de la flecha.

Entonces, tenemos a disposición una lista de las direcciones más utilizadas bajo el formato SYSEVAL:

Dirección (Hexadecimal)	Nombre de la Variable interna	Argumentos	Explicación
# 30794h	Verstring	→"HPHP48-"	Versión de la calculadora
# 3A328h	MakeStdLabel	\$ → grob	Realiza etiqueta de menú en forma gráfica
# 3A3ECh	MakeDirLabel	\$ → grob	Realiza etiqueta de Directorio en forma gráfica
# 3A38Ah	MakeBoxLabel	\$ → grob	Realiza etiqueta de activación en forma gráfica
# 3A44Eh	MakeInvLabel	\$ → grob	Realiza etiqueta invertida en forma gráfica
# 3A1FCh	DispMenu.1	→	Actualiza el menú actual
# 05F42h	Garbage	→	Actualiza la memoria remanente
# 353ABh	SYMB > IDS	symp→list	recupera variables de una expresión algebraica
# 40D25h	LockAlpha	→	Enciende teclado alfabético
# 40D39h	UnlockAlpha	→	Apaga teclado alfabético
# 3AA0Ah	1A/ LockA	→	Activa teclado alfabético una ocasión
# 44C31h	DoNewMatrix	→array	Llama al editor de matrices
# 44FE7h	DoOldMatrix	array→array'	Edita una matriz existente
# 1314Dh	TOADISP	→	Actualiza pantalla
# 15777h	Hidden Directory	→ ' '	Variable ocultas del sistema
# 05B15h	\$ > ID	\$ → id	Convierte Cadena en Objeto Global
# 05BE9h	ID > \$	id → \$	Convierte Objeto Global en cadena
# 41F65h	WaitForKey	→ <#Keypcode> <#Plane>	Posición de una tecla (Formato interno SYS-RPL)

PROGRAMA 26.

El siguiente programa reconoce las variable de una expresión algebraica de la forma $Y=F(x,y,z)$, que se guarda en la variable EQ; dichas variables se guardan en la variable VA. Las variables son reconocidas mediante el comando SYSEVAL **SYMB > IDS**; desde el entorno INPUT.

```
<< "Introduce la ecuación F(x)" { " ' ' α ALG } INPUT OBJ→
  STEQ EQ # 353ABh SYSEVAL 'VA' STO >>
```

Para el anterior ejemplo, es necesario que la ecuación sea introducida con precaución, debido a que al transformarse de una cadena de caracteres a una expresión algebraica, esta no puede tener errores de introducción, de ser así, la calculadora no ejecuta el programa debidamente.

PROGRAMA 27.

Se requiere que el programa tome los nombres de las variables de una ecuación de la forma $Y=F(x,y,z)$, y mediante una plantilla INFORM, se asignen valores a mencionada ecuación, para posteriormente, evaluarla. La ecuación se encuentra guardada en la variable EQ.

En el programa se ejecuta el comando SYSEVAL ID > \$, para llevar las variables al tipo caracteres, luego se ensambla la plantilla INFORM adecuadamente, como se explica en el capítulo I, apartado I. 1.2.. Se realizan verificaciones de la cantidad de variables para realizar dicho trabajo, y al mismo tiempo, verificar si estas variables son adecuadas para una sola plantilla de conversación INFORM.

```
<< 1 VA SIZE
    FOR i VA i GET
    # 05BE9h SYSEVAL ":"
    +
    "VERIFICAR UNIDADES"
    0 3 →LIST
    NEXT VA SIZE
    →LIST "VARIABLES"
    SWAP
    CASE
        IF VA SIZE 4 ≤
        THEN { }
        END
        IF VA SIZE 4 >
        VA SIZE 8 ≤ AND
        THEN { 2 }
        END { } { }
    IF INFORM
    THEN VA STO EQ →NUM "F(x)" →TAG
    END >>
```

PROGRAMA 28.

El siguiente trabajo requiere la introducción de una matriz, desde el entorno MATRIX-WRITER; dicha matriz se guarda en el archivo MATR. Se utiliza el comando SYSEVAL **DoNewMatrix**, y por su aplicabilidad, considerar que el trabajo es parte de un programa principal, y no puede parar su ejecución. Al mismo tiempo el programa verifica si se realizó la introducción de manera adecuada, para evitar algún error en la posterior ejecución.

```
<< ... DEPTH
      → n <<
          "Programa
           Listo para
           introducción
           de matriz" MSGBOX
          # 44C31h SYSEVAL
          DEPTH n IF ==
              THEN "Introducción
                   Cancelada" MSGBOX ...
              ELSE 'MATR' STO
              END
          >> ...
... >>
```

PROGRAMA 29.

El siguiente programa pregunta que si se desea editar la matriz existente guardada en la variable MATR, con menú de etiquetas invertidas personalizado, mediante el comando SYSEVAL **MakeInvLabel**; al mismo tiempo se utiliza el comando SYSEVAL **DoOldMatrix**, con la finalidad de editar la matriz existente.

```
<< ... "SI" # 3A44Eh SYSEVAL 1 →LIST
      "NO" # 3A44Eh SYSEVAL 1 →LIST
      2 →LIST TMENU CLLCD
      " Editar Matriz MATR ?" 4 DISP -1 WAIT
      IF 11.1 ==
      THEN MATR # 44FE7h SYSEVAL
      DROP 'MATR' STO
      END
      0 MENU ... >>
```

II.3. USO DEL COMANDO LIBEVAL

Con el mismo procedimiento para depurar los comandos SYSEVAL, se desarrollará los comandos LIBEVAL, y se explicará sus aplicaciones más prácticas. Al igual que SYSEVAL, es un comando de extrema potencialidad, y el mal uso podría causar la pérdida de memoria de la calculadora.

Los comandos LIBEVAL utilizados con más frecuencia son los siguientes:

Función LIBEVAL	Dirección Hexadecimal
Ventana con señal de advertencia	# B1000h
Menú de SOLVE POLYNOMIAL	# B402Ch
Menú de LINEAL SYSTEM	# B4033h
La ventana de CMD	# B2000h
Entrada a CHARS	# B2001h
Menú de MODOS (MODES)	# B41C1h
Menú CHOOSE de FLAGS (MODES)	# B41CFh
Entrada directa a MEMORY	# B41D7h
Entrada directa a SOLVE	# B4000h
Menú de SOLVE EQUATION	# B4001h
Menú de SOLVE DIFEQ EQUATION	# B4017h
Menú de TVM	# B4038h
Menú de PLOT	# B4045h
Menú de SYMBOLIC	# B4113h
Menú de Integrales	# B4114h
Menú de diferenciales	# B4122h
Menú de STAT	# B4175h
Menú de SINGLE-VAR STADISTIC	# B4176h
Menú de Frecuencias	# B417Dh
Menú de entrada de datos (FIT DATA)	# B417Fh
Menú de SUMARY STADISTICS	# B418Fh
Menú de I/O	# B4192h
Menú de Send to HP-48	# B4193h
Menú de Impresión	# B4197h
Menú de transferencia (TRANSFER)	# B41A8h
Menú de recepción de HP-HP	# B50FFh
Menú de TAYLOR	# B412Bh
Menú de ISOLATE VAR	# B412Dh
Menú de ecuación Cuadrática	# B4130h
Menú de manipulación de expresiones	# B4131h
Menú de la Aplicación TIME	# B4137h
Entrada de las alarmas	# B4138h
Set time and Date	# B415Bh
Catalogo de alarmas	# B416Eh

Como se explico en el principio del capitulo, los comandos LIVEBAL, sirven para facilitar el programa, pero no forman parte del programa. Los comandos LIBEVAL también podrían ser llamados los acceso directos de los diferentes menús constituyentes de las calculadoras Hewlett Packard.

PROGRAMA 30.

El siguiente programa, soluciona un sistema de dos ecuaciones implícitas, para el cálculo de acero requerido, en una viga de sección rectangular de Hormigón Armado, sometida a flexión simple; utilizando el comando LIBEVAL # B4001h. Considerar que el trabajo se realiza en plena ejecución de un programa, y que las ecuaciones se guardaron de la manera adecuada.

$$As = \frac{Mu}{\phi \times Fy \times (d - \frac{a}{2})} \qquad a = \frac{As \times Fy}{0.85 \times Fc \times b}$$

```
<<
...      '(As*Fy)/(0.85*Fc*b)' 'a' STO
          'As=Mu/(phi*Fy*(d-a/2))' STEQ
          # B4001h LIBEVAL
...      >>
```

PROGRAMA 31.

El siguiente programa, debe realizar una verificación de la configuración de la calculadora, para realizar trabajos con expresiones algebraicas o simbólicas.

```
<<      ...      "Desea verificar la configuración
                actual de la calculadora? S/N"
                { " " α } INPUT
                IF "S" SAME
                THEN # B41CFh LIBEVAL DROP
                END ... >>
```

PROGRAMA 32.

El programa en ejecución pide determinar si se desea realizar la derivación o integración de alguna expresión algebraica numérica o simbólica, dependiendo el caso, mediante el uso de los comandos LIBEVAL.

```
<<    ...    "Desea Derivar o Integrar?"  
        D/I" { " "  $\alpha$  } INPUT  
        IF "D"  
        THEN # B4122h LIBEVAL  
        ELSE # B4114h LIBEVAL  
        END ... >>
```

Con los ejemplos se tiene más clara la idea de que los comandos LIBEVAL, sirven potencialmente cuando se desea utilizar las plantillas o librerías propias de la calculadora de una manera que, en un programa normal no se podría ejecutar, debido a esto, LIBEVAL es un comando que sirve principalmente como acceso directo a las librerías internas contenidas en las calculadoras Hewlett Packard.

CAPITULO III: PROGRAMACION GRAFICA

III.1. OBJETIVO.

El objetivo del último capítulo, es de orientar al usuario, en la programación gráfica en calculadoras programables Hewlett Packard, es importante mencionar que, debido a lo extenso que es este tipo de programación, inicialmente se realizará una explicación de los comandos utilizados para la programación gráfica. Es importante mencionar que en la programación de entorno gráfico, los comandos no son independientes, como en los anteriores capítulos

Dentro la programación USER-RPL, se cuenta con dos directorios exclusivos de entorno gráfico, el directorio GROB, y el directorio PICT.

En principio, es importante definir la pantalla de la calculadora como una matriz de 131x64 elementos, 131 elementos por fila, y 64 elementos por columna. Cada elemento en este caso, será un PÍXEL, y el tratamiento que se da a estos es de dos maneras; la primera, considera una lista de dos números en formato Binario (Ej. { #131d #64d }), y es de fácil interpretación por parte del usuario; la segunda forma, es considerando que el píxel es compuesto por un par de coordenadas, similar al formato de número complejo, (Ej. (131,64)); estas interpretaciones no son las mismas, pero tienen relación o proporcionalidad.

Un objeto gráfico se representa con la palabra Graphic en la pila, esta palabra es seguida por dos números que indican la dimensión del gráfico (Ej. Graphic 131x64).

III.2. DIRECTORIO GROB

III.2.1. COMANDO →GROB

El uso del comando →GROB, obedece a la necesidad de crear gráficos, a partir de cadenas de caracteres, dicho comando requiere dos argumentos:

1. "Cadena de Caracteres"

Este argumento contiene la cadena de caracteres que se desea interpretar de forma grafica, es importante considerar la longitud de dicha cadena debido a que, en algunos casos, se puede realizar una excedencia de caracteres en pantalla, dando como resultado, una cadena gráfica de caracteres incompleta en pantalla.

2. # #

El segundo argumento es un número que define el tamaño de letra que tendrá la cadena de caracteres; el valor 1 como segundo argumento, especifica el tamaño de letra más pequeño; el valor 2, el tamaño medio, y el valor 3, el tamaño más grande de letra incorporado en la calculadora. Como excepción, se considera el valor 0, en el caso que se desee realizar la interpretación gráfica de una expresión algebraica; lógicamente en este caso el primer argumento, necesariamente tiene que ser una expresión algebraica o Simbólica.

PROGRAMA 33.

El siguiente programa crea una cadena de caracteres en formato gráfico, de tamaño 3.

```
<< "Hewlett Packard" 3 →GROB >>
```

Para visualizar este gráfico, se utiliza la secuencia << PICT STO PICTURE >>, comandos que pertenecen a un estudio mas adelante, pero que son necesarios por apreciación.

III.2.2. COMANDO BLANK

Este comando crea un objeto gráfico en la pila de la calculadora, y luego puede utilizarse para la edición o inclusión de elementos gráficos en dicho objeto gráfico inicial. Este comando requiere dos argumentos:

1. # AAFFFFd El primer argumento, en formato Entero Binario, este argumento indica la longitud en fila que tendrá el objeto gráfico.
2. # AAFFFFd El segundo argumento, en formato Entero Binario, este argumento indica la longitud en columna que tendrá el objeto gráfico.

PROGRAMA 34.

El trabajo que se requiere con el siguiente programa, es de crear un objeto grafico del tamaño de la pantalla, para luego incorporar este objeto, dentro el entorno gráfico incorporado en la calculadora.

```
<< # 131d #64 BLANK PICT STO >>
```

III.2.3. COMANDO GOR

El trabajo que realiza este comando, es de fundir un objeto gráfico sobre otro, por lo que es necesario determinar cuál es el objeto gráfico más grande, debido a que no se puede incluir un objeto más grande dentro de otro pequeño. Requiere de 3 Argumentos.

1. Graphic ### x ### Este es el objeto gráfico de mayor capacidad, al cual se añadirá otro gráfico de menor tamaño.

2. { # AFd # BFd } Las coordenadas de la esquina superior izquierda del objeto gráfico que se esta añadiendo, respecto al gráfico más grande. En este punto se debe considerar que, al especificar la esquina superior izquierda del gráfico pequeño, no se considera el extremo opuesto del grafico pequeño, por lo tanto, si el grafico que se añade sobrepasa el límite del gráfico de mayor tamaño, los PIXELS excedentes, son descartados.

3. Graphic ## x ## El gráfico que se va a fundir en el primero.

PROGRAMA 35.

El siguiente programa, realiza la fusión de dos objetos gráficos creados por el comando BLANK, el primer gráfico es de 65x32 PIXELS, y se lo funde sobre un gráfico del tamaño de la pantalla, tratando que este quede centrado. En este caso se toma el negativo del grafico de 65x32 PIXELS, para su apreciación.

```
<<    # 131d # 64d BLANK  
      { # 33d # 16d }  
      # 65d #32d BLANK NEG  
      GOR PICT STO PICTURE >>
```

III.2.4. COMANDO GXOR

Este comando es muy parecido al comando GOR, con la diferencia que en vez de fundir los dos gráficos, se realiza un intercambio de píxeles, dicho en palabras más sencillas, invierte los píxeles, si estos están ocupados, o los ocupa, si estos están libres.

PROGRAMA 36.

Un ejemplo muy claro del uso de este comando, es el siguiente:

Se tiene un objeto gráfico totalmente oscuro del tamaño de la pantalla, esto se logra, tomando el negativo de un objeto gráfico creado con el comando BLANK; y se desea insertar la cadena "Hewlett Packard" en medio, con el tamaño de letra 1.

```
<<   # 131d # 64d BLANK NEG
      "HEWLETT PACKARD" 1 →GROB
      { # 25d # 25d } SWAP GXOR
      PICT STO PICTURE >>
```

III.2.5. COMANDO SUB

El comando SUB, tiene la función de extraer una parte de un gráfico, a partir de los extremos opuestos de dicha fracción. Requiere tres argumentos:

1. Graphic ## x ## El objeto gráfico del cual se requiere extraer una fracción de gráfico.
2. { # AFd # BFd } Esquina superior izquierda de la fracción del gráfico que se requiere.
3. { # BFd # CFd } Esquina inferior derecha de la fracción del gráfico que se requiere.

PROGRAMA 37.

El siguiente programa requiere extraer la palabra TEXTO de la cadena en formato gráfico "TEXTO DE PROGRAMACIÓN", dicha cadena esta con el tamaño de letra 2.

```
<<   "TEXTO DE PROGRAMACION" 2 →GROB
      PICT STO PICTURE
      PICT RCL { # 0d # 0d }
      { # 30d # 8d } SUB
      PICT STO PICTURE >>
```

III.2.6. COMANDO REPL

Este comando es parecido a los comandos GOR y GXOR que se estudio anteriormente, con la diferencia que en este caso, sobrepone al objeto gráfico que se esta insertando, eliminando totalmente los píxeles utilizados en el gráfico original. Requiere 3 argumentos:

1. Graphic ### x ### Este es el objeto gráfico de mayor capacidad, al cual se sobrepodrá otro gráfico de menor tamaño.
2. { # AFd # BFd } Las coordenadas de la esquina superior izquierda del objeto gráfico que se esta añadiendo, respecto al gráfico más grande. En este punto se debe considerar que, al especificar la esquina superior izquierda del gráfico pequeño, no se considera el extremo opuesto del grafico pequeño, por lo tanto, si el grafico que se añade sobrepasa el límite del gráfico de mayor tamaño, los PIXELS excedentes, son descartados.
3. Graphic ## x ## El gráfico que se va a sobrepone en el primero.

PROGRAMA 38.

El siguiente programa, sobrepone el texto "PROGRAMACION HP", DE TAMAÑO 3, sobre un gráfico creado anticipadamente.

```
<<    # 131d # 64d BLANK NEG  
      "PROGRAMACION HP" 3 →GROB  
      { # 20d # 10d } SWAP REPL  
      PICT STO PICTURE >>
```

III.2.7. COMANDO →LCD

El comando →LCD tiene un uso muy particular; como indica el comando, lleva a un objeto gráfico a la pantalla, pero no se tiene la opción de editar el gráfico, como al utilizar el comando PICTURE.

III.2.8. COMANDO LCD→

Al contrario del comando LCD→, este lleva a la pantalla actual de la calculadora, a formato gráfico. El uso adecuado de los comandos LCD→ y →LCD, depende de la habilidad o conveniencia del usuario

PROGRAMA 39.

El siguiente programa, simula encontrarse en la pantalla actual de la calculadora, y envía un saludo durante un segundo.

```
<< LCD→  
"HOLA TU" 1 →GROB  
{ # 51d # 17d } SWAP REPL  
→LCD 1 WAIT >>
```

III.2.9. COMANDO SIZE

La función de este comando es determinar el tamaño de un objeto gráfico, en números Enteros Binarios, indicando la longitud de filas y columnas. El uso de este comando es frecuente cuando requiere definir los límites o dimensiones de un objeto gráfico que se incluye a otro. Se realizara una aclaración de este comando de una manera mas completa, posteriormente, debido a que se requiere el estudio de otros comandos adicionales.

III.2.10. COMANDO ANIMATE

El comando ANIMATE, sirve para realizar una animación o secuencia de gráficos, no necesariamente del mismo tamaño. Requiere como argumentos, una determinada cantidad de gráficos, que se quieren interpretar como una animación, y una lista que contenga 4 elementos:

{ ##	{ # AFd # BFd }	##	## }
Cantidad de gráficos	Posición inicial	Duración entre Imágenes [sg.]	Repeticiones

Cantidad de gráficos, indica el número total de gráficos que forman parte de la animación.

La posición inicial indica las coordenadas en las que se ejecutará la secuencia gráfica.

La duración entre imágenes especifica el tiempo que transcurre antes de realizar un cambio de imagen, en segundos.

El número de repeticiones indica cuantos ciclos se realizaran con la animación; si este campo toma el valor de cero, se está indicando que el número de ciclos es indefinido.

PROGRAMA 40.

El siguiente programa realiza la animación de cuatro imágenes, mostrando una secuencia de textos "TEXTO DE PROGRAMACION", con un tiempo entre imágenes de 0.5 segundos, tratando que la animación se encuentre en el centro de la pantalla.

```
<<  # 131d # 64d
      BLANK PICT STO
      # 76d # 16d BLANK
      NEG → A
      <<  A DUP "TEXTO" 1 →GROB
          { # 28d # 5d } SWAP
          GXOR A "DE" 2 →GROB
          { # 32d # 4d } SWAP
          GXOR A "PROGRAMACION" 3 →GROB
          { 4 { # 27d # 24d } 0.5 0 } ANIMATE
          5 DROPN >>
      >>
```

III.3. DIRECTORIO PICT

Después de haber desarrollado el contenido del directorio GROB, es conveniente determinar la relación que existe entre los números Enteros Binarios, y los números complejos.

Cuando se crea un gráfico, tiene sus dimensiones en números Enteros Binarios, pero al mismo tiempo, esta en función a un rango de números, positivos o negativos dependiendo de los límites propuestos o por defecto. Antes de desarrollar el contenido del directorio PICT, es importante ampliar con los siguientes comandos, que pertenecen al menú 81 (PLOT):

XRNG, es el comando que define cual es el rango numérico que se desea tener en el eje horizontal, requiere de dos argumentos, el primero define el límite inferior, y el segundo, el límite superior; como se explicó, estos valores, están dentro el rango de los números reales

YRNG, es el comando que define cual es el rango numérico que se desea tener en el eje vertical, requiere de dos argumentos, el primero define el límite inferior, y el segundo, el límite superior; estos valores, tienen que estar dentro el rango de los números reales

Con estos conceptos, se puede decir que se tiene una relación entre el formato de número Entero Binario, y el número complejo, por ejemplo, si se define un gráfico de 131x64 píxeles, y se decide tener un rango de datos o rango numérico de 65x32, estamos determinando que un dato numérico corresponde a una posición en formato Entero Binario; es decir, un par de coordenadas (0,0), corresponde a la posición { # 0d # 63d }, las coordenadas (65,32), corresponde a la posición en formato Entero binario { # 130d # 0d }.

Por esta razón, los comandos de graficación que se incluyen dentro en el directorio PICT, y que son desarrollados a continuación, pueden ser utilizados con los dos formatos distintos.

III.3.1. COMANDO PICT

El comando PICT, se interpreta como una variable propia de la memoria de las calculadoras, en la que se guarda el objeto grafico actual, y puede ser utilizado, editado, borrado, etc, tal como se trata con una variable creada por el usuario.

III.3.2. COMANDO PDIM

El comando PDIM, tiene la función de crear un objeto gráfico, al igual que el comando BLANK (Apartado III.2.2), con la diferencia de que el grafico no aparece en la pila, automáticamente se incorpora en la variable PICT propia de la calculadora. Dicho de otra forma, se efectúa un dimensionado del objeto gráfico

que se va a realizar posteriormente. Otro uso es que, con los comandos de número complejo, se efectúa el rango de valores numéricos que se desea utilizar. Requiere dos argumentos, para cada caso:

Caso 1: Tamaño del Gráfico

1. # AAFFFd El primer argumento, en formato Entero Binario, este argumento indica la longitud en fila que tendrá la variable PICT.
2. # AAFFFd El segundo argumento, en formato Entero Binario, este argumento indica la longitud en columna que tendrá la variable PICT.

Caso 1: Rango de valores en X, Y

1. (## , ##) El primer argumento, en formato de número complejo. Este argumento indica el límite inferior de valores en el eje X y el eje Y, respectivamente .
2. (## , ##) El segundo argumento, en formato de número complejo. Indica el límite superior de valores en el eje X y el eje Y, respectivamente

III.3.3. COMANDO LINE

Este comando sirve para realizar un trazo o línea, a partir de sus dos extremos, estos están en formato de números Enteros Binarios. Requiere dos argumentos:

1. { # AFd # BFd } Coordenada en formato Entero Binario de uno de los extremos de la línea que se va a generar.
2. { # BFd # CFd } Coordenada en formato Entero Binario del otro extremo de la línea que se desea generar.

La idea de realizar el trazo, a partir de los valores numéricos, difiere solo en los argumentos necesarios para ejecutar este comando, lo que si es importante, es determinar cuál es el rango de datos apropiado o correcto.

1. (## , ##) Coordenada en formato de número complejo. Un extremo de la línea que se va a generar.
2. (## , ##) Coordenada en formato de número complejo. Segundo extremo de la línea que se va a generar.

PROGRAMA 41.

El siguiente programa, realiza una generación de líneas dentro un gráfico guardado en la variable PICT, la generación es de líneas verticales. Se logra visualizar el proceso por la secuencia <<... { # 0d # 0d } PVIEW ...>> y se define un tiempo de dos segundos antes que se deje de visualizar el gráfico.

```
<<   # 131d # 64d PDIM
      { # 0d # 0d } PVIEW
      15 115 FOR i
      i R→B # 10d 2 →LIST
      i R→B # 50d 2 →LIST
      LINE 5 STEP 2 WAIT >>
```

PROGRAMA 42.

Si en el mismo tamaño de gráfico se decide realizar trazos en un rango numérico de 0 a 200 en el eje X, y de 0 a 200 para el eje Y, el programa sería como el que se tiene a continuación:

```
<<   # 131d # 64d PDIM
      0 200 XRNG
      0 200 YRNG
      { # 0d # 0d } PVIEW
      20 180 FOR i
      i 20 R→C
      i 180 R→C
      LINE 10 STEP 2 WAIT >>
```

III.3.4. COMANDO TLINE

Al igual que el comando LINE, TLINE realiza un trazo a partir de dos coordenadas en formato Entero Binario o número complejo, con la diferencia que, con la misma idea del comando GXOR, si la línea intercepta con algún píxel encendido, este se apaga; en el caso que este se encuentre apagado, se enciende. Sus argumentos:

1. { # AFd # BFd } Coordenada en formato Entero Binario de uno de los extremos de la línea que se va a generar.

2. { # BFd # CFd } Coordenada en formato Entero Binario del otro extremo de la línea que se desea generar.

PROGRAMA 43.

El siguiente programa, realiza una generación de líneas dentro un gráfico guardado en la variable PICT, que contiene líneas verticales. La generación es de líneas horizontales. Se define un tiempo de 2 segundos, después de terminado el trabajo, para visualizar el gráfico.

```
<<   { # 0d # 0d } PVIEW
      15 45 FOR i
      # 10d i R→B 2 →LIST
      # 120d i R→B 2 →LIST
      TLINE 3 STEP 2 WAIT >>
```

El procedimiento para trabajar con números complejos es similar, es importante recalcar que, para obtener resultados satisfactorios, se determinen los rangos numéricos.

III.3.5. COMANDO BOX

El comando BOX, genera un recuadro a partir de dos datos, que son los vértices opuestos; dichos vértices pueden estar en formato Entero Binario o en número complejo.

PROGRAMA 44.

Se requiere generar 5 cuadrados de diferentes tamaños dentro un gráfico de 131x64 píxeles.

```
<<   # 131d # 64d PDIM
      { # 0d # 0d } PVIEW
      10 50 FOR i
      130 2 / i 2 / - R→B
      64  2 / i 2 / - R→B
      2  →LIST
      130 2 / i 2 / + R→B
      64  2 / i 2 / + R→B
      2  →LIST BOX 10 STEP
      2  WAIT >>
```

PROGRAMA 45.

En un rango numérico de $\pm 100x \pm 100$, se requiere generar 10 rectángulos de diferentes tamaños dentro un gráfico de 131x64 píxeles, y luego convertir el gráfico en su negativo. Se da un tiempo de 2 segundos para apreciar el gráfico.

```
<<   # 131d # 64d PDIM
      -100 100 XRNG -100 100 YRNG
      { # 0d # 0d } PVIEW
      10 100 FOR j
      j NEG j R→C
      j j NEG R→C
      BOX 10 STEP
      2 WAIT PICT NEG >>
```

III.3.6. COMANDO ARC

El comando ARC determina el arco de una circunferencia en sentido contrario a las agujas del reloj, a partir de cuatro datos o argumentos, en los diferentes formatos explicados:

1. (x,y)
ó
{ # AFd # BFd }
El centro de la circunferencia, en formato complejo, considerando el rango de datos numéricos.
El centro de la circunferencia, en formato de número Entero Binario, considerando las dimensiones del gráfico.
2. # #
ó
AFd
El radio en el eje X, formato de números complejos.
Radio considerando el formato de números Enteros Binarios.
3. $\theta 1$
Angulo inicial, soporta cualquiera de los dos formatos de graficación.
4. $\theta 2$
Angulo final, soporta cualquiera de los dos formatos de graficación.

Los argumentos 3 y 4, están en función al tipo de medición angular configurado en la calculadora, es decir, si se encuentra en formato de Grados, Radianes o Gradientes.

PROGRAMA 46.

El programa realiza la graficación de un arco de circunferencia de radio 5, desde 0° hasta 90°, en un gráfico del tamaño de la pantalla, con un rango numérico de 20x20. El centro de la circunferencia está en el origen.

```
<<   DEG
      # 131d # 64d PDIM
      (0,0) (20,20) PDIM
      { # 0d # 0d } PVIEW
      (0,0) 5 0 90 ARC
      2 WAIT >>
```

III.3.7. COMANDO PIXON

El comando PIXON ocupa el píxel que se desea encender, con un argumento; la coordenada si se trabaja con el plano numérico definido, ó la coordenada en formato Entero Binario que se requiere ocupar del gráfico actual.

PROGRAMA 47.

El programa realiza una generación de píxeles en toda la pantalla de la calculadora, dejando un píxel libre entre píxeles.

```
<<   # 131d # 64d PDIM
      { # 0d # 0d } PVIEW
      0 131 FOR i
      0 64 FOR j
      i j 2 →LIST R→B
      PIXON
      2 STEP
      2 STEP 2 WAIT >>
```

III.3.8. COMANDO PIXOFF

El comando PIXOFF es el inverso al comando PIXON, realiza el trabajo de apagar un píxel encendido, también requiere de un argumento, numero complejo, o coordenadas en formato Entero Binario.

Adicionalmente, se cuenta con el comando **PIX?**, que realiza la verificación si el píxel actual se encuentra encendido, mediante los operadores lógicos predeterminados en la calculadora, que fueron explicados en el capítulo I.

PROGRAMA 48.

El siguiente programa se ejecuta sobre el gráfico creado en el anterior programa, realiza una verificación si el píxel actual está encendido, de ser así, lo apaga; si el píxel está apagado, lo enciende, es decir, invierte el gráfico actual. El trabajo se realiza con el comando **PIX?** y se define un rango numérico de 131x64.

```
<<   { # 0d # 0d } PVIEW
      0 131 XRNG 0 64 YRNG
      0 131 FOR i
      0 64 FOR j
      i j R→C → A
      <<   IF A PIX?
          THEN A PIXOFF
          ELSE A PIXON
          END >>
      NEXT NEXT 2 WAIT >>
```

III.3.9. COMANDO PVIEW

El comando **PVIEW**, que ya fue utilizado anteriormente por razones demostrativas, realiza las funciones de indicar cual es la coordenada inicial en la que se desea ver el gráfico, o animación gráfico. Este comando requiere de un argumento en formato Entero Binario, o en formato de número complejo.

1. { # AFd # BFd } Coordenada inicial en pantalla del gráfico o animación.
 ó
 (#, #) Coordenada inicial en número complejo.

Cuando se trata de visualizar un solo gráfico, y dejar la pantalla en ese estado, el argumento que requiere el comando **PVIEW**, es una lista vacía, ({ }), y su uso es muy particular. Si **PVIEW** se ejecuta de esta manera, centra el gráfico y es mostrado como en el entorno **PICTURE**, pero no se tiene el cursor activado. En este caso, el gráfico mostrado persiste hasta que se presione la tecla **CANCEL**.

III.3.10. COMANDO PX→C

Este comando tiene la función de interpretar una coordenada en formato Entero Binario, como un par de coordenadas en el plano, es decir, en número complejo. Es un comando muy útil, cuando se desea determinar la relación o correspondencia entre los dos formatos de graficación utilizados por la calculadora. Como se puede notar, requiere como argumento una lista de números Enteros Binarios, que componen una posición en el entorno gráfico actual.

III.3.11. COMANDO C→PX

Este comando es inverso al anterior, tiene la función de interpretar una coordenada en formato de número complejo, como una posición en números Enteros Binarios. Como argumento requiere un número complejo, para determinar una posición correspondiente en formato Entero Binario.

Estos comandos, así como muchos otros que fueron estudiados, pueden ser utilizados de acuerdo a la conveniencia o habilidad del programador, los diferentes usos y aplicaciones adecuadas, harán que un programa realice un determinado trabajo, de la manera más óptima, ocupando una menor cantidad de memoria, o reduciendo pasos intermedios, para realizar los mismos trabajos. El programador cuenta con las herramientas, y la depuración para tener el uso más adecuado esta en función a la práctica que se realice y experiencias que se tengan al realizar diferentes programas.

- FIN DEL CAPITULO III -

PROGRAMA 49.

El siguiente programa realiza una verificación del estado de memoria actual de la calculadora, verifica que tipo de calculadora se utiliza, y muestra cuál es la memoria utilizada en librerías, variables de usuario, y memoria libre.



```

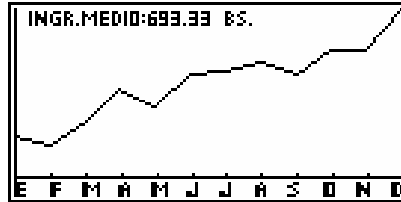
<<
<< # 131d # 64d
PDIM { # 0d # 0d }
PVIEW PICT DUP
{ # 24d # 1d } ←MO 2
→GROB REPL
{ # 72d # 3d } "(MEMORIA)"
1 →GROB REPL { # 5d
# 15d } { # 125d
# 20d } BOX ←A ←CM
/ 100 * 0 RND → ←X
<< 5 ←X 100 /
120 * 5 + 0 RND
FOR i i R→B
# 16d 2 →LIST i R→B
# 19d 2 →LIST LINE
NEXT ←B ←CM /
100 * 0 RND → ←Y
<< ←X 100 /
120 * 5 + DUP ←Y
100 / 120 * 5 + +
FOR i PICT
i R→B # 15d 2 →LIST
GROB 2 6 302010201030
REPL 2
STEP PICT { # 10d # 31d }
"VARIABLES:" ←A +
" BYTES (" + ←X +
"%)" + 1 →GROB REPL
PICT { # 10d # 39d }
"LIBRERIAS:" ←B +
" BYTES (" + ←Y +
"%)" + 1 →GROB REPL
PICT { # 10d # 47d }
"BYTES LIBRES:"
←MEM + "(" + 100
←X ←Y + - + "%)" +

```

```
1 →GROB REPL
  >>
  >>
  >> PATH HOME VARS
    0 PVARs 0 0 → ←EJE
    ←PAT ←VAR ←LIB ←MEm
    ←A ←B
  << 1 ←VAR SIZE
  FOR i ←VAR i
    GET BYTES ←A + '←A'
    STO DROP
  NEXT 1 ←LIB
  SIZE
  FOR i ←LIB i
    GET RCL BYTES ←B +
    '←B' STO DROP
  NEXT
  CASE ←A ←B ←MEm
  + + 32000 > ←A ←B
  ←MEm + + 128000 ≤
  AND
  THEN "HP48GX"
  128000 → ←MO ←CM
  << ←EJE EVAL
  >>
  END ←A ←B +
  ←MEm + 32000 <
  THEN "HP48G"
  32000 → ←MO ←CM
  << ←EJE EVAL
  >>
  END ←A ←B
  ←MEm 128000 >
  THEN "HP49G"
  256000 → ←MO ←CM
  << ←EJE EVAL
  >>
  END
  END ←PAT EVAL { } PVIEW
  >>
  >>
```

PROGRAMA 50.

El programa muestra de forma gráfica el estado comercial de una determinada actividad financiera, y muestra un promedio apreciativo.



```

<< # 131d # 64d PDIM
{ "Enero" "Febrero"
"Marzo" "Abril"
"Mayo" "Junio"
"Julio" "Agosto"
"Septiembre"
"Octubre"
"Noviembre"
"Diciembre" } { } 0
1 → MESES AC VE C
<< 1 12
  FOR i
    "Ingreso para el mes
de "
MESES i GET +
": (Bs.)" + { " " }
INPUT OBJ→ DUP VE >
  IF
    THEN DUP 'VE'
STO
  END AC + 'AC'
STO
  NEXT AC REVLIST
'AC' STO { # 0d
# 0d } PVIEW { # 0d
# 57d } { # 131d
# 57d } LINE { # 0d
# 0d } { # 0d # 64d
} LINE { "E" "F"
"M" "A" "M" "J" "J"
"A" "S" "O" "N" "D"

```

```
} 0 → A B
  << 0 131
    FOR i PICT i
R→B # 59d 2 →LIST A
B 1 + DUP 'B' STO
GET 1 →GROB REPL
PICT i 1 + R→B
# 59d 2 →LIST A B
GET 1 →GROB REPL
11.5454545455
  STEP 0 131
    FOR i i R→B
# 56d 2 →LIST PIXON
11.9090909091
  STEP 0 131
XRNG 0 VE 10 + YRNG
0 131
  FOR i i AC C
GET 10 + R→C i
11.9090909091 + AC
C 1 + DUP 'C' STO
GET 10 + R→C LINE
11.9090909091
  STEP PICT {
# 5d # 1d }
"INGR.MEDIO:" AC
ΣLIST 12 / 2 RND +
" BS." + 1 →GROB
REPL { } PVIEW
  >>
  >>
  >>
```

REFERENCIAS.

La Bibliografía y referencias que fueron revisadas para la elaboración de este texto, fue la siguiente:

1. Guía del Usuario Serie HP 48G (Hewlett Packard)
2. Manual del Usuario Avanzado Series HP 48G (Hewlett Packard)
3. Manual del Usuario Serie HP 49 (Versión Internet) (Hewlett Packard)
4. Apuntes, ejemplos de programación propios del autor.
5. www.hpcalc.org - Archivos SYSTRANS, Base de datos SYSEVAL

Para mayor información, respecto a este texto, dudas, comentarios o sugerencias, por favor dirigirse a:

Gustavo Alejandro Marka Saravia
Ingeniero Civil
Teléfonos: (4) 4298585 - 71721091
E-mail: guzzal@hotmail.com
Cochabamba - Bolivia