Curso de Programación User RPL

Por: Luis M. Gallardo D.
lmgd@cantv.net
http://www.angelfire.com/ok/hp48g

Tabla de contenidos

1. Introducción.

El User-Rpl es el lenguaje al que tenemos acceso desde la calculadora, y es la forma mas elemental de programarla, pero la primera que debemos aprender. A continuación se presenta el contenido de este tema, el cual tiene los comandos mas usados para la programación de la calculadora.

2. ¿Qué es un Programa?.

Un programa, para la calculadora, es todo aquello que este entre << >>. Un ejemplo de programa puede ser: << 1 2 + >>, el cual coloca en la pila de datos el numero 1, el 2 y luego los suma.
Para editar cualquier programa hecho bajo User-Rpl solo se tiene que colocar en pila y pulsar "EDIT". Para guardarlo se coloca en pila y luego se coloca el nombre con el que se quiera grabar y se pulsa "STO".

```
3:  << 1 2 + >>
2:  'Nombre'
1:  STO
```

Para ejecutarlo se puede colocar en la pila y pulsar "EVAL", o llamarlo por su nombre.

3. Declaración de Variables.

Dentro de los programas es común llamar objetos u otros programas con variables, que pueden ser Estáticas, Locales o Globales.

3.1. Variables Estáticas:

Las Variables Estáticas son guardadas en la memoria de usuario (RAM) y son independientes del programa que las llama. La declaración de una variable de este tipo es equivalente a guardar el contenido de la variable en un directorio con un nombre, como si se tratase de un programa.

3.2. Variables Locales:

Las variables Locales son las que solo se ejecuta en un programa y solo son reconocidas en este. Para declarar variables locales se sigue la siguiente sintaxis:

```
<< obj1 obj2 → V1 V2 >>
```

Las variables locales solo serán reconocidas en el bloque que fueron declaradas, o en los bloques contenidos dentro del bloque en que se declararon. A veces se puede tener varios bloques que generen confusión, por lo que se tiene que tener cuidado con el rango de acción de las variables Locales, ya que puede que no sean reconocidas en todo el programa.
EJEMPLO:

```
<< →  V1 V2
  << V1 V2 * V1 V2 + → M S  (1)
    << M S / >> (2)
  >>  V2 V1 *
>>
```

En el ejemplo anterior las variables V1 y V2 solo son reconocidas en los bloques (1) y (2), aunque se hayan declarado al principio.

3.3. Variables Globales:

Las variables Globales son aquellas que se declaran en un programa principal y pueden ser reconocidas, en cualquier momento, desde cualquier bloque de instrucción o desde cualquier programa que se llame y se ejecute en el programa principal. Para declarar una variable de este estilo se sigue la sintaxis:

```
<< obj1 obj2 → ←V1 ←V2 >>.
```

Con esta declaración no se debe tomar precaución con el rango de acción de una variable, ya que es reconocida en todo el programa, pero una declaración de este tipo hace que los programas ocupen mayor espacio en memoria, por lo que debe de usarse cuando sea "ESTRICTAMENTE NECESARIO".

4. Funciones de Comparación o Test.

Las funciones de comparación son aquellas que comparan argumentos, devolviendo un 1 (true = Verdadero) o un 0 (false = Falso) de acuerdo a lo que se este preguntando. Estas funciones se describen en la siguiente tabla:

Función de Comparación	Descripción
SAME	Pregunta si dos objetos son iguales.
= =	Pregunta si dos objetos son iguales.
≠	Pregunta si dos objetos son distintos.
<	Pregunta si el obj2 es menor al obj1.
>	Pregunta si el obj2 es mayor al obj1.
≥	Pregunta si el obj2 es mayor o igual al obj1.
≤	Pregunta si el obj2 es menor o igual al obj1.

Como se puede apreciar SAME y == cumplen la misma función.

5. Funciones Lógicas.

Las funciones Lógicas son aquellas que permiten dar a conocer la relación entre dos condiciones. Estas funciones son el AND y el OR.

AND: Esta función permite dar a conocer si se cumplen Simultáneamente dos condiciones, y responde a la siguiente tabla de la verdad:

Condición #1	Condición #2	Respuesta
0	0	0
0	1	0
1	0	0
1	0	1

OR: Con esta función se puede conocer si Una de las dos condiciones se cumple, y esta dado por la tabla de la verdad:

Condición #1	Condición #2	Respuesta
0	0	0
0	1	1
1	0	1
1	1	1

Adicionalmente se tiene otra función Lógica (**NOT**), pero que no es para saber el estado de dos condiciones, sino que sirve para invertir el estado de una condición; y esta dada por la siguiente tabla de la verdad:

Condición	Condición Negada
0	1
1	0

Esta función no devuelve ningún tipo de argumento o valor.

6. Estructuras Condicionales.

Una de las características de los lenguajes de programación modernos es que son estructurados, es decir, que responden a una estructura secuencial, por lo tienen un inicio, un desarrollo (que son las instrucciones que se ejecutan una tras otra) y un final. Los programas hechos en estos tipos de lenguajes pueden diseñarse y probarse en frío previamente, mediante algoritmos que resuelvan la situación presentada.
Las Estructuras Condicionales permiten ejecutar una acción a través de la toma de decisiones ante una o varias condiciones dadas.

6.1. IF...THEN...ELSE...END:

Esta estructura se conoce como partición o "si" binario, ya que:

SI se cumple una **CONDICIÓN**
ENTONCES se realiza una **ACCIÓN**
SINO se ejecuta otra **ACCIÓN DIFERENTE**
FIN

Esta estructura presenta una variante interesante, que es la "partición simple", que es:

SI se cumple una **CONDICIÓN**
ENTONCES se realiza una **ACCIÓN**
FIN

Hay que resaltar que si se cumple la condición se ejecuta la acción, pero de resto no se realiza nada.
EJEMPLO DE PARTICIÓN BINARIA:

```
<< -
   IF 0 >
   THEN "POSITIVO"
   ELSE "NEGATIVO"
   END
>>
```

El programa que hicimos resta dos números en la pila, pregunta si el resultado es positivo o negativo, y lo indica en cada caso.
EJEMPLO DE PARTICIÓN SIMPLE:

```
<<
   IF ==
   THEN "Elementos iguales"
   END
>>
```

Este otro programa pregunta si dos objetos colocados en pila son iguales, generando un mensaje en caso afirmativo, en caso contrario no se ejecuta ninguna acción.

6.2. CASE...END:

Con las estructuras "IF" solo se puede tomar una decisión de entre máximo dos condiciones, por lo que si se quiere tomar una decisión de varias se tendría que colocar varios "IF" anidados unos dentro de otros o de forma consecutiva. Esto es engorroso, y de esta necesidad surgió la estructura "CASE", que sigue la siguiente sintaxis:

```
<<
   CASE CONDICION_1
      THEN  ACCION_1
      END CONDICION_2
      THEN  ACCION_2
      END
                .
                .
                .
          CONDICION_n
      THEN  ACCION_n
      END ACCION_POR_DEFECTO
   END
>>
```

NOTA:
El "CASE" solo ejecuta una condición y es la primera que encuentre como verdadera. Si no encuentra ninguna ejecuta la acción por defecto.
EJEMPLO CON "IF"

```
<< → N
   <<
      IF N 0 ==
      THEN "ES IGUAL A CERO"
      ELSE
        IF N 0 >
        THEN "ES POSITIVO"
        ELSE "ES NEGATIVO"
        END
      END
   >>
>>
```

MISMO EJEMPLO CON "CASE"

```
<< → N
   <<
      CASE N 0 ==
        THEN "ES IGUAL A CERO"
        END N 0 >
        THEN "ES POSITIVO"
        END "ES NEGATIVO"
      END
```

```
    >>
>>
```

Hay que destacar que el "CASE" pregunta primero si N es igual a cero, luego si es positivo y si no es asume la ultima condición como negativa.

7. Estructuras Iterativas.

Las estructuras iterativas son aquellas que nos permiten repetir varias veces un proceso. Estas estructuras son de gran importancia conocerlas, ya que cuando se programa su uso en la resolución de un problema es frecuente.

7.1. START...NEXT:

Una de las estructuras soportadas por la calculadora es el START, la cual es una de las estructuras no condicionales, es decir, es independiente de cualquier condición y se efectúa un numero finito de veces. La sintaxis que sigue es la siguiente:

```
<< 1 #veces
   START ACCIÓN
   NEXT
>>
```

En este tipo de estructura se le indica el número de veces a repetir el proceso con un 1 y el número de veces a repetir, al comienzo del ciclo. El proceso a ejecutar viene seguido después del comando START, y el final del ciclo lo determina el comando NEXT.
EJEMPLO:

Supongamos que queramos calcular el factorial de un numero, por ejemplo 5! = 5x4x3x2x1, lo resolveríamos con el siguiente programa.

```
<< 1 → N AUX
   << 1 N 1 –
   START N AUX * 'AUX' STO 'N'
1 STO-
   NEXT
   >>
>>
```

Este programa efectúa el factorial, no solo del número cinco sino, de cualquier número que este en la pila. Si al verlo surge alguna duda, recomiendo correrlo con el debbuger o depurador de la calculadora. Esto se hace colocando el programa en la pila y pulsando la secuencia de teclas PRG NXT RUN DBUG STT. De esta forma se vera como funciona el programa paso a paso. Otra forma de ver si un programa está correcto es la corrida en frío, que se hace siguiendo el comportamiento del programa en una hoja de papel. Recomiendo que se copie el programa en el papel, y las variables que intervienen en el mismo en una tabla, a medida que se va siguiendo el comportamiento del programa se ve el estado de cada variable.

7.2. DO...UNTIL...END:

Este tipo de estructura pertenece al bloque de los ciclos condicionales, ya que dependen de la evaluación de una condición dada. El DO ejecuta un proceso y luego pregunta por una condición,

para ver si ejecuta nuevamente el proceso; y así sucesivamente. La estructura que del DO es la siguiente:

```
      <<
         DO PROCESO
         UNTIL CONDICIÓN
         END
      >>
```

EJEMPLO:

```
      << 1 → I
         <<
            DO  I 1 + 'I' 2 STO+
            UNTIL I 100 ==
            END
         >>
      >>
```

En el ejemplo anterior realizamos un programa que genera los primeros 50 primeros números pares.

7.3. WHILE...REPEAT...END:

El WHILE tiene como característica fundamental el preguntar una condición antes de ejecutar el proceso. El WHILE sigue la sintaxis:

```
      <<
         WHILE CONDICIÓN
         REPEAT PROCESO
         END
      >>
```

La condición puede ser el resultado de evaluar una situación dada o puede ser un contador, que no es mas que una variable que lleva el control del numero de veces que se repite el proceso.
EJEMPLO:

```
      << -5
          WHILE DUP 0 <
          REPEAT 1 +
          END
      >>
```

En el programa anterior se pregunta si se cumple la condición de que el numero que se toma en pila es menor que cero, mientras que es menor se repite el proceso de sumar uno al número de la pila. Hay que notar que para preguntar sobre la condición (si es menor a cero) primero se duplica el numero en al pila con el comando DUP.

8. Contadores y Banderas en la Iteración.

Las estructuras iterativas condicionales (DO y WHILE) repiten un proceso si ocurre una condición específica. Si se quiere repetir un proceso un número determinado de veces podemos usar lo que se denomina contadores, que son variables se incrementa a medida que se ejecutan esos procesos.
EJEMPLO:

```
<< 5 → I
   <<
      WHILE I 0 >
      REPEAT I 'I' 1 STO-
      END
   >>
>>
```

El programa coloca los números del 1 al 5 en pila, usando un ciclo que es controlado por la variable I, la cual es un contador.
Las banderas se usa en la iteración para salirse de un ciclo sin llegar al final del mismo. Como ejemplo se va a tomar el mismo programa anterior, pero colocándole una bandera.
EJEMPLO:

```
<< 5 → I
   <<
      WHILE I 0 >
      REPEAT I 'I' 1 STO-
        IF I 3 ==
        THEN 0 'I' STO
        END
      END
   >>
>>
```

La bandera está en el IF, ya que al llegar I a el valor de 3 lo coloca con valor de cero, luego el WHILE analizar la condición y confirmará que I no es mayor que cero, así termina el ciclo.

9. Control del Teclado.

Un programa en ejecución puede tomar una decisión de acuerdo a una tecla que es oprimida. Esto es posible debido a que el programa reconoce cualquier tecla, ya que cada botón del teclado está identificado. Para reconocer un botón, se asume que el teclado de la calculadora es una de matriz, que tendría la siguiente forma:

Identificación de las teclas					
11.x	12.x	13.x	14.x	15.x	16.x
21.x	22.x	23.x	24.x	25.x	26.x
31.x	32.x	33.x	34.x	35.x	36.x
41.x	42.x	43.x	44.x	45.x	46.x
51.x		52.x	53.x	54.x	55.x
61.x	62.x	63.x	64.x	65.x	-
71.x	72.x	73.x	74.x	75.x	-
81.x	82.x	83.x	84.x	85.x	-
ON	92.x	93.x	94.x	95.x	-

Donde "x" puede tomar 7 valores (0,1, 2, 3, 4, 5 ó 6). El valor de x depende de si se usa una tecla de asignación de función, es decir, si se pulsó antes la tecla con la flechita morada, la flechita verde, la tecla de asignación de carácter (alfa) ó una combinación de estas tres. Si se ordena en una tabla resulta:

Tecla de asignación de función	Tecla en la HP	Valor
Ninguna	---	0 ó 1
Desplazamiento a la Derecha		2
Desplazamiento a la Izquierda		3
Alfa		4
Alfa + Desplazamiento a la Derecha		5
Alfa + Desplazamiento a la Izquierda		6

Entonces si tenemos como identificación de una tecla el valor de 82.2 significa que pulsamos primero la tecla de asignación de función morada y luego pulsamos el número 1. (ver tabla de asignación de teclas, y la de funciones).
Los comandos usados para el reconocimiento del teclado son "KEY" Y "WAIT", los cuales generan una respuesta con el formato explicado anteriormente.

9.1. KEY.

Este comando es usado para identificar a cualquier tecla en el momento de ejecución de un programa. La forma como reconoce una tecla es de manera instantánea, es decir, si se ejecuta devuelve el número de identificación de la tecla que se estaba pulsando en ese instante. Si no se está pulsando ninguna devuelve el valor cero.
EJEMPLO:
Como este comando devuelve el valor de la tecla que se esté pulsando se incluirá en un bloque de iteración para ver su efecto, pero es válido ejecutar << KEY >> para ver su funcionamiento.

```
<<
   DO  KEY DUP
   UNTIL 0 ≠
   END
>>
```

Este programa ejecuta la instrucción KEY hasta que se pulsa una tecla. Si el valor devuelto es 0 se sigue repitiendo el ciclo.

9.2. WAIT.

A diferencia del KEY, el comando WAIT devuelve el número de identificación de la tecla que se halla pulsado en un período de tiempo especificado. Si no se pulsa en ese tiempo ninguna tecla no aparecerá nada en la pila de la calculadora.
EJEMPLO:

```
<< 3 WAIT >>
```

La ventaja del WAIT sobre el KEY es que se puede utilizar para esperar que se pulse una tecla en un período indeterminado de tiempo sin usar una estructura iterativa adicional. Esto se logra colocando como período de tiempo el valor de cero al comando WAIT.
EJEMPLO:

```
<< 0 WAIT >>
```

Una posible desventaja de usar WAIT es que cuando se oprima la tecla ON, con este último ejemplo, se devolverá el valor cero en vez de la posición de la tecla correspondiente.

10. Entrada de Datos.

Durante la ejecución de un programa en la calculadora nos puede interesar tomar datos desde el teclado. Para hacer esto posible se pueden usar los comandos INPUT o INFORM. El comando INPUT es apropiado cuando se trabaja con programas que utilicen pocas variables, mientras que el INFORM nos permite trabajar más fácilmente con múltiples variables y dar una descripción de lo que significa cada una.

10.1. INPUT.

Este comando nos permite tomar datos desde el teclado de una forma sencilla. Es aconsejable utilizarlo cuando se está trabajado pocas variables, dado a la forma como se presenta la información, pero de igual forma se puede trabajar con múltiples variables. La sintaxis que caracteriza este comando para una sola variable es la siguiente:

```
<< "Título" ":Nombre_variable:"
INPUT
>>
```

NOTA:
Es importante que se coloque el nombre de la variable entre ": :" para que sea reconocida por el INPUT.

Si se quiere trabajar con más variables, la sintaxis es la siguiente:

```
<< "Título" {
"
:Nombre_variable_1:
:Nombre_variable_2:
:Nombre_variable_3:"
{1 0} } INPUT
>>
```

Para colocar varias variables se colocan en una lista los nombres de las mismas, el último elemento de la lista "{ 1 0 }" especifica la fila.
EJEMPLO:

```
<< "Suma de 3 Números" {
":N1:
:N2:
:N3 "  {1 0} } INPUT
OBJ→ + + "Suma" →TAG
>>
```

En este ejemplo se toman tres variables (N1, N2 y N3) y se llevan a formato numérico a través del comando "OBJ→" , luego se suman y se etiqueta el resultado, con el comando "→TAG", bajo el nombre "Suma".

10.2. INFORM.

El INFORM es una forma más elaborada de tomar los datos desde el teclado, ya que este comando permite brindar información al usuario de las variables que se estén trabajando. La forma de utilizar este comando es la siguiente:

```
<< "Título"
{ {"Nombre_variable_1:" "Comentario_1" Tipo_de_dato }
{"Nombre_variable_2:" "Comentario_2" Tipo_de_dato }
{"Nombre_variable_3:" "Comentario_3" Tipo_de_dato }}
{ #Columnas #Espacio}
{ Valor_Reset_1 Valor_Reset_2 Valor_Reset_3 }
{ Valor_defoult_1 Valor_defoult_2 Valor_defoult_3 }
INFORM
>>
```

Primero se especifica el título de la plantilla. Luego se coloca el nombre de cada variable con su respectivo comentario y el tipo de dato que es aceptado para esa variable (los tipos de datos viene en el manual de usuario de la calculadora). Después se determina el número de columnas por pantalla que deben de aparecer, así como la distancia del campo a llenar y el nombre de la variable. A continuación se coloca una serie de valores que aparecerán cuando se borre el contenido de uno o todos los campos; y por último se especifica los valores por defecto que han a de aparecer cuando se ejecute el comando INFORM.

Todo esto parece un poco complicado pero no lo es, ya que muchos de estos parámetro se pueden obviar. Como ejemplo colocaré un programa considerando todos los parámetro, otro programa donde obviaré algunos y uno donde no consideraré a ningún parámetro.
EJEMPLO 1 (Considerando todos los parámetros).

```
<< "Agenda" { {"Nombre:"
"COLOQUE UN NOMBRE" 2 } {
"Teléfono:" "NUMERO DE TELEFONO"
} {"E-mail:"
"DIRECCION DE CORREO" 2 } }{ 1
0 } { Carlos 5769985
"carlos@fecel.net" } { Luis
123456 "hp48g@mailcity.com" }
INFORM
>>
```

En este ejemplo se consideraron todos los parámetros. Se especificaron comentarios y tipos de datos en cada variable, numero de columnas y espacio entre las variables y su campo, valores que aparecen al borrar los campos y los valores por defecto
EJEMPLO 2 (Obviando algunos parámetros).

```
<< "Agenda" { {"Nombre:"
"COLOQUE UN NOMBRE"  } {
"Teléfono:" "NUMERO DE TELEFONO"
} {"E-mail:"
"DIRECCION DE CORREO" } } { } {
} { } INFORM
```

```
>>
```

En este otro ejemplo solo se consideraron los parámetro de los comentarios, y se ignoraron los restantes. Al no especificar ningún tipo de dato en cada variable, se está dando la libertad de colocar cualquier tipo de dato (número, carácter, lista, etc.).
EJEMPLO 3 (Obviando todos los parámetros).

```
<< "Agenda" { "Nombre:"
"Teléfono" "E-mail:"} { } { } {
} INFORM
>>
```

En este último ejemplo no se consideró ningún parámetro. Esta es la forma más elemental, y la que ocupa menos espacio en memoria, de trabajar con el INFORM. En esta básica del INFORM, los nombres de cada variable se colocan en una sola lista, a diferencia de los ejemplos anteriores donde cada variable estaba en una lista diferente (esto debido a que cada variable contenía un comentario y tipo de dato asociado).

11. Captura de Errores.

Una gran ventaja que posee la HP-48 es la posibilidad de trabajar los errores como condiciones, es decir, que si ocurre un determinado error en la ejecución de algún programa la calculadora puede tomar una decisión a raíz de ese evento, sin detener la ejecución de dicho programa.

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

El IFERR responde a la necesidad de poder controlar errores en la ejecución de programas. Esta estructura tiene gran semejanza con el IF y funciona de la siguiente manera:

SI ocurre un ERROR **(CONDICIÓN)**
ENTONCES se realiza una **ACCIÓN**
SINO se ejecuta otra ACCIÓN **DIFERENTE**
FIN

La sintaxis para esta estructura es la siguiente:

```
<<
   IFERR Bloque_de_instrucción o Programa
   THEN Acción
   ELSE Acción_Diferente
   END
>>
```

El "IFERR" nos garantiza que si ocurre un error en la ejecución de un programa la calculadora realizará una acción, sino realizará otra. Esta estructura se puede trabajar como partición binaria (sintaxis anterior) o como partición simple, para realizar una acción si se genera un error o no hacer nada en caso contrario. La sintaxis para la partición simple es:

```
<<
   IFERR Bloque_de_instrucción o Programa
   THEN Acción
   END
>>
```

EJEMPLO:

```
<<
   IFERR  "Suma de 3 Números" {
":N1:
:N2:
:N3 "{1 0} } INPUT
OBJ→ + + "Suma" →TAG
   THEN CLEAR "ERROR DE EJECUCION" DOERR
   END
>>
```

En este ejemplo se uso el programa de "Suma de 3 Números", del ejemplo de INPUT, para probar el funcionamiento del IFERR. Si ocurre un error en el programa de "Suma de 3 Números", bloque de instrucciones entre el IFERR y el THEN, se ejecutan las acciones de limpiar la pila (CLEAR) y generar un mensaje de error con el comando DOERR. Para que se produzca un error en programa solo hay que dejar de introducirle las variables y darle a ENTER, cuando el programa trate sumar los valores se encontrará con que no existen argumentos, y así se producirá el error.
Con esta sencilla sintaxis del IFERR se puede controlar el funcionamiento de cualquier programa o conjunto de instrucciones.

12. Salida de Datos.

Cuando elaboramos un programa puede que nos interese presentar datos de salida para informar el resultado de alguna operación o proceso. Para realizar esto la calculadora dispone de comandos destinados para tal fin, y su uso depende de cómo se desee presentar los datos. Dentro de esos comandos se encuentran: DISP, MSGBOX Y PVIEW.

12.1 DISP.

Este comando permite mostrar líneas de textos (no mayores de 22 caracteres) en la pantalla de la calculadora. Si la línea de texto se pasa de los 22 caracteres, la calculadora sustituye los caracteres siguientes por "...". Como el tamaño de cada letra es igual al tamaño real de las letras en la calculadora, solo se dispone de 7 líneas por pantalla. La sintaxis que se sigue para mostrar líneas de textos es la siguiente:

```
<< "LINEA DE TEXTO" #NUMERO_DE_LINEA DISP
>>
```

EJEMPLO:

```
<< CLLCD "HOLA" 4 DISP -1 FREEZE
>>
```

Con el comando DISP se pueden mostrar líneas de texto en la pantalla de la calculadora, pero si la pantalla contenía algo previo la línea se escribirá sobre eso. Para evitar esto existe una instrucción que limpia la pantalla: CLLCD.
Otro detalle que tiene el comando DISP, es que su ejecución ocurre de forma inmediata, por lo que si se quiere ver su efecto se tendrá que usar en conjunto con el comando FREEZE, el cual congela la pantalla en un tiempo determinado. Si se coloca como parámetro de tiempo el valor de -1 al comando FREEZE, este congelará la pantalla de la calculadora hasta que se oprima otra tecla.
Para ver el efecto de DISP también se puede usar el comando WAIT, colocándole a este último un parámetro de tiempo ó el valor cero para un tiempo de espera indeterminado (hasta que se pulse otra tecla).

EJEMPLO CON WAIT:

```
<< CLLCD "HOLA" 4 DISP 0 WAIT
DROP
>>
```

EL comando DROP se usa para eliminar el resultado que devuelve el WAIT al oprimir una tecla.
Con el comando DISP también se pueden mostrar varias líneas de textos al mismo tiempo, solo se tienen que colocar en una lista las líneas de texto y en otra lista sus respectivos números de líneas. Las sintaxis es la siguiente:

```
<< {"LINEA DE TEXTO 1" "LINEA DE TEXTO 2"}
```

```
{#NUMERO_DE_LINEA_1 #NUMERO_DE_LINEA_2 } DISP >>
```

EJEMPLO:

```
<< CLLCD { "HOLA" "AMIGO" }  { 2 4
} DISP -1 FREEZE
>>
```

Con esta sintaxis se pueden mostrar hasta 7 líneas de texto (de hasta 22 caracteres) por pantalla al mismo tiempo. Cada línea de texto debe tener asociado un número de línea en pantalla, para que pueda aparecer.

12.2. MSGBOX.

El comando MSGBOX permite crear cajas de diálogos en la calculadora. Estas cajas solo tienen como objeto el informar algún mensaje, por lo que solo se cierran oprimiendo la tecla de menú "OK". Para crear una caja de dialogo solo se tiene que usar esta sintaxis:

```
<< "TEXTO" MSGBOX
>>
```

EJEMPLO:

```
<< CLLCD "HOLA" MSGBOX
>>
```

El comando CLLCD limpia la pantalla para que de fondo no salga nada. Si se omite, aparecerá de fondo lo que estaba antes de ejecutar MSGBOX.

12.3. FONDOS.

Si se desea crear un efecto que impacte, se puede usar en conjunto el comando DISP y el comando MSGBOX. Combinándolos se puede obtener un fondo para una caja de diálogo.

EJEMPLO:

```
<< "@@@@@@@@@@@@@@@@@@@@@@" {1 2
3 4 5 6 7} DISP
"ATRAS ESTA UN FONDO" MSGBOX
>>
```

Como fondo se puede utilizar cualquier combinación de caracteres de la calculadora.

13. Salida de Datos Con Gráficos.

Otra forma de mostrar datos e información en la calculadora es usando gráficos. Cualquier dato se puede convertir (sea del tipo que sea) al formato de gráficos de la calculadora, es decir a un GROB, y luego mostrarlo por pantalla. Los comandos usados para convertir y mostrar gráficos por pantalla son →GROB, PICT, REPL, PVIEW y →LCD entre otros.

13.1. ® GROB.

Este comando nos permite convertir cualquier tipo de objeto a un grob de la HP. Para usar este comando se usa la siguiente sintaxis:

```
<< Objeto_en_Pila #Tamaño_de_la_letra →GROB >>
```

Donde Objeto_en_Pila se refiere a que se tiene que tener un objeto en la pila para ejecutar el comando y #Tamaño_de_la_letra indica que tipo de letra queremos usar. El número a usar por tamaño de letra de resume en la siguiente tabla:

Numero de Fuente	Tipo de Letra
1	Grob Letra Pequeña
2	Grob Letra Mediana
3	Grob Letra Grande
0	Grob de Ecuaciones

El tipo de Grob 0 nos permite crear grobs de ecuaciones, los cuales se mostrarán como si fuese desplegado por el Equation Writer de la HP.

EJEMPLO:

```
<< "Saludos Amigos" 2 →GROB
>>
```

Este ejemplo convierte el string en un grob de fuente 2, para tratarse como un objeto gráfico.

13.2. PICT.

Este comando nos permite cargar un grob entero en la pantalla gráfica de la HP, o traer un grob entero desde la pantalla gráfica a la pila. Para cargar un grob en la calculadora desde la pila se usa la sintaxis << Grob PICT STO >> donde Grob indica que se tiene que tener un grob en la pila. Para traer un grob a la pila se usa esta otra sintaxis << PICT RCL >>.
EJEMPLOS PARA CARGAR UN GROB

```
1. << "Saludos Amigos" 1 →GROB PICT STO >>
2. << "Saludos Amigos" 2 →GROB PICT STO >>
```

```
3. << 'A+B=C' 0 →GROB PICT STO >>
```

El ejemplo 1 convierte el string en un grob de fuente pequeña y el ejemplo 2 en fuente mediana y se cargan en la pantalla gráfica, pero el ejemplo 3 convierte el objeto simbólico en un grob de ecuaciones para cargase como si fuese el Equation Writer de la HP.

EJEMPLO PARA TRAER UN GROB A LA PILA
Para traer un grob a la pila solo se tiene que usar

```
<< PICT RCL>>.
```

13.3. PVIEW.

De nada nos sirve colocar un grob en la pantalla gráfica si no lo podemos ver. Para visualizar un grob cargado en pantalla se usa el comando PVIEW. La sintaxis usada para que se pueda ejecutar es la siguiente:

```
<< -40 CF { # X # Y } PVIEW
>>

      ó

<< -40 CF{}PVIEW
>>
```

Donde - 40 CF desactiva la actualización del reloj en la calculadora, y { #X #Y } corresponden a las coordenadas de la esquina superior izquierda de la pantalla, que comúnmente son { #0d #0d }. Si se quiere ver el efecto que produce colocar las coordenadas dentro de la lista se debe usar en conjunto con el comando WAIT, de la siguiente forma: `<< { #X #Y } PVIEW N WAIT >>`, donde N es el tiempo que esperará el comando WAIT. Si se coloca el valor de 0 ó -1, se mantendrá a la calculadora en estado de espera y el grob cargado en la pantalla gráfica permanecerá mostrándose hasta que se pulse una nueva tecla. Con está sintaxis no se puede usar los desplazamientos de pantalla, por lo que si el tamaño del grob es mayor a lo máximo desplegable por la calculadora (131x64) no se podrá apreciar en su totalidad, ya que se terminará de mostrar al pulsar una nueva tecla.
Si no se especifican las coordenadas de la esquina superior izquierda de la pantalla dentro de la lista que precede al comando, como está en la segunda sintaxis, se mostrará el grob en la pantalla hasta que se pulse CANCEL, sin necesidad de usar el comando WAIT. Si se utiliza esta sintaxis se tiene la posibilidad de usar los desplazamiento de pantallas, y en consecuencia ver los grob que sobre pasen lo máximo desplegable por la pantalla de la calculadora (131x64).
La primera sintaxis es muy útil si se quiere mostrar una imagen que se actualizará con la ejecución de un programa, mientras que la segunda es útil para cuando se quiere mostrar grobs que no varían, como es el caso de una ecuación o una fórmula.
EJEMPLOS:

```
1.    << -40 CF
      "SALUDOS AMIGOS, ¿COMO ESTAN?" 1
      →GROB PICT STO { # 0d # 0d }
```

```
      PVIEW 0 WAIT DROP
      >>

2.    << -40 CF
      "Saludos Amigos" ¿Cómo están?  2
      →GROB PICT STO { # 0d # 0d }
      PVIEW 0 WAIT DROP
      >>

3.    << -40 CF 'A+B=C' 0 →GROB PICT
      STO {}PVIEW
      >>
```

13.4. REPL.

Bueno, ya podemos cargar un grob a la pantalla y verlo, pero ahora queremos que el grob mostrado actualmente varíe agregándole otro grob. Para esto la HP posee un comando llamado REPL. La sintaxis usada es la siguiente:

```
<< PICT { # X # Y } Grob_en_la_pila REPL >>
```

Donde {#X #Y} representa la coordenada donde aparecerá el grob en pantalla. En esta coordenada se colocará la esquina superior izquierda del nuevo grob. Grob_en_la_pila se refiere a que debe de existir un grob en la pila para colocarlo en la pantalla.
El comando REPL con está sintaxis no devuelve valores a la pila, ya que inserta directamente el grob en la pantalla actual.
EJEMPLO:

```
<< -40 CF
"SALUDOS AMIGOS, ¿COMO ESTAN?" 1
→GROB PICT STO { # 0d # 0d }
PVIEW 1 WAIT PICT { # 10d # 10d }
"ESTAMOS BIEN" 1 →GROB REPL 0
WAIT DROP
>>
```

REPL también se puede usar para insertar un subgrob en una sección de un grob que este en la pila. Para hacer esto se usa el comando de la siguiente forma:

```
<< Grob_destino { # X # Y } Grob_en_la_pila REPL
>>
```

Grob_destino, es el grob donde queremos insertar el subgrob y {#X #Y} corresponden a las coordenadas donde se colocará. El comando usado con esta sintaxis si devuelve un valor, que es el grob resultante de colocar el Grob_en_la_pila al Grob_destino en las coordenadas {#X #Y}.

EJEMPLO:

```
<< -40 CF
"SALUDOS AMIGOS, ¿COMO ESTAN?" 1
```

```
→GROB PICT STO PICT RCL { # 10d
# 10d } "ESTAMOS BIEN" 1 →GROB
REPL PICT STO { # 0d # 0d }
PVIEW 0 WAIT DROP
>>
```

El comando REPL no solo sirve para insertar un grob al grob mostrado en la pantalla actualmente o a un grob que esté en la pila, sino que también sirve para insertar un elemento a una cadena de String, a una lista, a un vector o a una matriz. Para usar el comando REPL con estas intenciones, consulte el manual de la calculadora.

13.5. SUB.

Si lo deseamos podemos usar una sección rectangular de un grob, extrayéndola de un grob mostrado en pantalla o que este en la pila. Para hacer esto se usa el comando SUB con la siguiente sintaxis:

```
<< Grob_en_la_pila
{ # X1 # Y1}{ # X2 # Y2} SUB
>>
```

Grob_en_la_pila indica nuevamente que se tiene que tener un grob en la pila para usar el comando SUB, {#X1 #Y1} indica la coordenada de la esquina superior izquierda de la sección rectangular, y {#X2 #Y2}indica la esquina inferior derecha de la sección rectangular a extraer.
Todo esto se realizó con un grob que ya estaba en la pila, pero si se quiere extraer algo de un grob que se esté mostrando en pantalla se tendrá que traer el grob a la pila y seguir los mismos pasos. Entonces la sintaxis es la misma, solo hay que agregarle dos comandos adicionales:

```
<< PICT RCL Grob_en_la_pila
{ # X1 # Y1}{ # X2 # Y2 } SUB
>>
```

EJEMPLO:

```
<< -40 CF
"SALUDOS AMIGOS, ¿COMO ESTAN?" 1
→GROB PICT STO { #0d  # 0d }
PVIEW 1 WAIT PICT { # 10d # 10d }
"ESTAMOS BIEN" 1 →GROB REPL PICT
RCL { # 10d # 10d } { # 70d # 20d
} SUB 0 WAIT DROP
>>
```

Al terminar de ejecutarse este programa, en la pila quedará un grob de 61x11, que contiene la frase "ESTAMOS BIEN". Si se desea se puede cargar a la pantalla y verse posteriormente.

13.6. ® LCD y LCD ® .

Estos comandos sirven para mostrar un grob que esté en la pila (→LCD) o llevar a un grob lo que se esté mostrando en la pantalla actualmente (LCD→). La sintaxis para los comandos son las siguientes:

```
<< Grob_en_la_pila →LCD
>>

<< LCD→
>>
```

Cuando se trabaja con el comando →LCD hay que tener en cuenta que se seguirán mostrando las etiquetas de menú actuales, además que no se puede trabajar con los comandos REPL y SUB. El comando →LCD no es afectado por la actualización del reloj en la pantalla, por lo que no es necesario desactivar el reloj.

Al utilizarse el comando LCD→ se tiene que considerar que solo llevará a la pila lo que se esté mostrando en la pantalla de la calculadora, y el tamaño correspondiente de ese grob es de 131x64.

EJEMPLO DE →LCD:

```
<< "SALUDOS AMIGOS, ¿COMO ESTAN?"
1 →GROB →LCD
>>
```

A este último ejemplo se le debe añadir un FREZEE o un WAIT para que perdure su efecto.