UNIVERSIDAD TECNICA DE ORURO
FACULTAD NACIONAL DE INGENIERIA
CARRERA DE INGENIERIA CIVIL

Curso de Programación en User-RPL

Orientado en calculadoras HEWLETT PACKARD
Series 48/49: HP48G/G+/GX/GII, HP49G/G+

Expositor:

Univ. Sandro Choque Martínez

Oruro, abril de 2005

CONTENIDOS

I. Introducción a la programación.

1. Introducción.

El User RPL se puede catalogar como un lenguaje de alto nivel (Los lenguajes de alto nivel son normalmente fáciles de aprender porque están formados por elementos de lenguajes naturales a nuestra comprensión humana), este es similar a muchos otro lenguajes que existen como Pascal, Visual Basic y otros. Además del User RPL (RPL del Usuario) existen el System RPL (RPL del sistema) y el ML (Machine Language: Lenguaje Máquina; algunas veces se le usa (¿incorrectamente?) queriendo decir lenguaje ensamblador. El User RPL basado en la manipulación de la pila; es un lenguaje estructurado (tipo de programación que produce código con un flujo limpio, un diseño claro y un cierto grado de modularidad o de estructura jerárquica), dicho de otro modo cada programa tiene una entrada (principio del programa) y una salida (fin del programa). Se puede sacar provecho a la estructuración del lenguaje creando programas de bloque constitutivo; cada programa constitutivo puede permanecer solo o funcionar como una subrutina de un programa mayor: así, desde un programa se puede llamar a otro sin preocuparnos de nada ya que una vez finalizado el programa llamado se devolverá el flujo al programa principal. A pesar de ser lento, es muy poderoso y fácil de usar, lo único que se debe saber es algo de algoritmia, lógica y aprender los comandos de codificación (sintaxis de programación). Los comandos son muy buenos, con ellos puede hacerse prácticamente todo lo imaginable: manejar cadenas y gráficos (incluso sprites, muy útil para los juegos), realizar operaciones de cualquier tipo y acceder a la infinidad de funciones preprogramadas de que dispone la calculadora.

2. User RPL.

RPL es el lenguaje más simple para programar en la HP, el cual no es más que un lenguaje de escritura. El acrónimo de RPL: Reverse Polish Lisp es decir Lisp Polaco Inverso. Originalmente no iba a hacer de conocimiento público este término, y se le mencionaba simplemente como RPL. Más tarde la HP trató de hacerlo significar ROM Procedural Language es decir Lenguaje de Procedimientos del ROM, pero Reverse Polish Lisp persistió.

Algunas personas lo llaman Reverse Polish Language es decir Lenguaje Polaco Inverso, o Reverse Polish Logic es decir Lógica Polaca Inversa, pero ninguno de estos es el nombre oficial.

3. Orígenes de User RPL.

Los orígenes del RPL se remota a aquellos años de 1984, donde se inicio un proyecto en la división de Hewlett-Packard en Corvallis para desarrollar un nuevo software de un sistema operativo para el desarrollo de una línea de calculadoras y soportar una nueva generación de hardware y software.

Anteriormente todas las calculadoras HP se implementaron enteramente en lenguaje ensamblador, un proceso que se iba haciendo cada vez más pesado e ineficiente. Los objetivos para el nuevo sistema operativo fueron los siguientes:

- Proporcionar control de la ejecución y manejo de la memoria, incluyendo memoria conectable.
- Proporcionar un lenguaje de programación para un rápido desarrollo de aplicaciones y obtención de prototipos.
- Soportar una variedad de calculadoras de negocio y técnicas.
- Ejecución idéntica en RAM y ROM.
- Minimizar el uso de memoria especialmente RAM.
- Ser transportable a varias CPU's;
- Ser extensible.
- Soportar operaciones de matemática simbólica.

Se tuvieron en cuenta varios lenguajes y sistemas operativos ya existentes pero ninguno cumplía con todos los objetivos del diseño; por consiguiente se desarrolló un nuevo sistema, el cual mezcla la interpretación entrelazada de Forth con el enfoque funcional de Lisp.

El sistema operativo resultante, conocido de modo no oficial como RPL (de Reverse-Polish Lisp), hizo su primera aparición pública en junio de 1986 en la calculadora Business Consultant 18C. Más tarde, RPL ha sido la base de las calculadoras HP-17B, HP-19B, HP-27S, HP-28C y HP-28S y HP 48S y HP 48SX. La HP-17B, 18C y la 19B se diseñaron para aplicaciones de negocios; ellas y la calculadora científica HP-27S ofrecen una lógica de cálculo "algebraica" y el sistema operativo subyacente es invisible para el usuario.

Las familias HP 28/HP 48 de calculadoras científicas usan una lógica RPN y muchas de la facilidades del sistema operativo están disponibles directamente como comandos.

4. Menú de comandos de la Pila.

En la siguiente tabla se describen los comandos programables que manipula la pila.

COMANDO	DESCRIPCION
CLEAR	Borra la Pila
DEPTH	Devuelve el número del objetos de la pila
DUP	Duplica el objeto del nivel 1
DUPDUP*	Duplica un objeto del nivel 1 dos veces
DUP2	Duplica 2 objetos del nivel 1y 2
DUPN	Duplica n objetos en la pila comenzando por el nivel 2 (n esta en el nivel 1)
DROP	Borra el primer objeto de la pila
DROP2	Borra los objetos de los niveles 1 y 2
DROPN	Borra los primeros objetos n+1 de la pila (n está en el nivel 1)

NDUPN*	Duplican veces el objeto del nivel 2 (n debe estar en el nivel 1)
NIP*	Borra el objeto del nivel 2 de la pila
OVER	Devuelve una copia del objeto del nivel 2
PICK	Copia el objeto del nivel n+1 al nivel 1 (n esta en el nivel 1)
PICK3*	Copia el objeto del nivel 3 al nivel 1 de la pila
ROLL	Desplaza el objeto del nivel n+1 (n esta en el nivel 1)
ROLLD	Desplaza hacia abajo una parte de la pila entre el nivel 2 y el nivel n+1 (n esta en el nivel 1)
ROT	Hace girar los 3 primeros objeto = 3 ROLL
SWAP	Invierte dos objetos de la pila del nivel 1 y 2
UNROT*	Desplaza el objeto del nivel 1 al nivel 3 de la pila

* Comandos disponibles solo en la HP49

Estos comandos están disponibles desde el menú de comandos (STACK):

HP 48 SERIES (STACK)

HP 49 SERIES

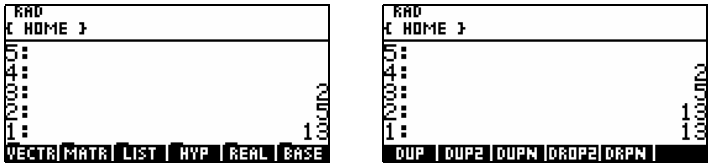
También puedes ingresar al menú de comandos (STACK), escribiendo en la pila: 73 MENU

Ejemplo 01: *Duplicar un objeto ubicado en el nivel 1 de de la pila utilizando DUP (este también puede ser reemplazado por ENTER)*

HP 48 SERIES

Paso 1: 13 ENTER

Paso 2: NXT



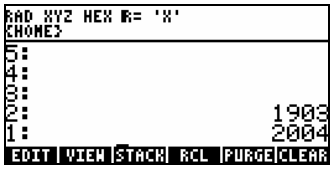
HP 49 SERIES

Paso 1: 13 ENTER

Paso 2:

Ejemplo 02: *Para cambiar los objetos de los niveles 1 y 2 utilice SWAP*

Paso 1: 1903 [SPC] 2004 [ENTER]
Paso 2: [TOOL] [STACK] [SWAP]



5. Modo Algebraico y RPN.

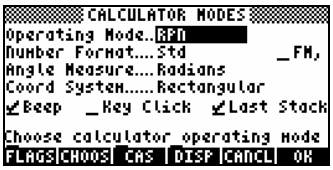
Existen dos formas de trabajo en la pila RPN y el algebraico que a continuación detallamos, y por ende el modo que usaremos para el curso es el modo RPN.

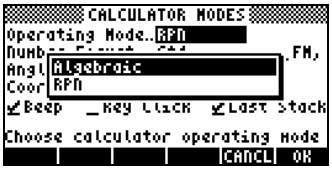
5.1. Modo Algebraico.

En el modo algebraico los cálculos se realizan introduciendo los argumentos después del comando, es decir que en la mayoría de las operaciones significa introducir números, funciones y operadores en el mismo orden en los que resolvemos una operación en papel.

Por ejemplo para encontrar el Seno de 0.25 en modo algebraico, pulse [SIN] el comando y luego el argumento (.25)

En la HP49 el modo algebraico es el modo por defecto. Para fijar el modo algebraico se sigue los siguientes pasos: Pulse [MODE] luego [CHOOS] elegir Algebraic y [OK]

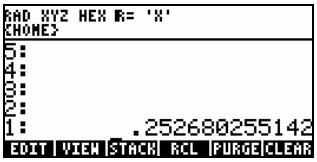




5.2. Modo RPN.

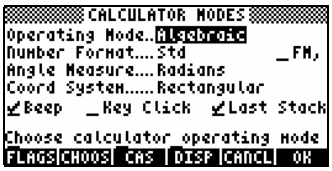
Jan Lukasiewicz escribió un libro de lógica formal en 1951 mostrando que las expresiones matemáticas podían especificarse sin paréntesis, colocando los operadores antes (Polish Notation; notación polaca) o después (Reverse Polish Notation; notación polaca inversa) de los operandos. En este modo generalmente se ingresa un argumento antes del comando.

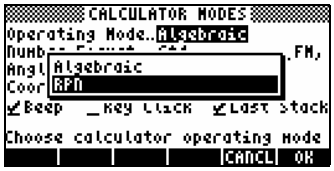
Por ejemplo para encontrar el ArcoSeno de 0.25 en modo RPN primero se ingresa el argumento 0.25 luego el comando [←] [SIN] (ASIN) la respuesta aparece sin necesidad de pulsar [ENTER] otra vez.



En el modo RPN, los resultados de cálculos anteriores se listan tal como están en modo algebraico. Sin embargo, son sólo los resultados (y no los cálculos) esta lista de resultados anteriores (y otros objetos se denomina stack (pila) y cada elemento de la misma esta numerado.

Para fijar el modo RPN se sigue los siguientes pasos: Pulse MODE luego elegir RPN y





6. Funciones de comparación o Test.

Las funciones de comparación son aquellas que comparan argumentos, devolviendo `1` (TRUE: Verdadero) si es verdadero ó `0` (FALSE: Falso) si es falso de acuerdo a lo que se este preguntando. Estas funciones se describen en la siguiente Tabla:

FUNCION DE COMPARACION	DESCRIPCION
SAME	Pregunta si dos objetos son iguales
==	Pregunta si dos objetos son iguales
≠	Pregunta si dos objetos son distintos
<	Pregunta si el objeto 2 es menor al objeto 1
>	Pregunta si el objeto 2 es mayor al objeto 1
≤	Pregunta si el objeto 2 es menor o igual al objeto 1
≥	Pregunta si el objeto 2 es mayor o igual al objeto 1

También puedes ingresar al menú de comandos de comparación o test, escribiendo en la pila: `32 MENU`

Ejemplo 03: *Compare si dos números puestos en la pila son iguales.*

Paso 1: 5 SPC 3 ENTER
Paso 2: PRG

Se ha utilizado la función == comparando si los objetos puestos en la pila son iguales, el resultado nos da 0 (FALSO), entonces los números no son iguales.

7. Funciones lógicas.

Las funciones lógicas son aquellas que permiten dar a conocer la relación entre dos condiciones, estas toman uno o dos argumentos (objetos) de la pila (cualquier real distinto de cero es tomado como uno (verdadero), sólo el cero es considerado como falso). Las funciones son:

AND: Devuelve verdadero (1) si ambos argumentos son verdaderos
OR: Verdadero si al menos uno es verdadero
XOR: Verdadero si uno y sólo uno es verdadero
NOT: Siempre devuelve el inverso lógico

Condición 1	Condición 2	AND	OR	XOR
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

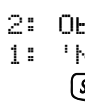
NOT: Siempre devuelve el inverso lógico

Condición	NOT
1	1
0	1

8. Almacenar, Recuperar, Borrar Objetos.

8.1. Almacenar (Guardar) un objeto

Después de entrar un objeto en la pila, este se lo puede almacenar en una variable asignándole un nombre. Pueden utilizarse nombres descriptivos para las variables. Un nombre puede tener una sola letra o hasta 127. Los nombres de variables no podrán ser idénticos a los nombres de comandos ni comenzar por un número. Para almacenar un objeto se sigue la siguiente sintaxis:



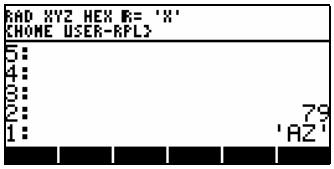
Ejemplos 04: *Almacenar el número 79 (constante) bajo el nombre 'AZ'*

HP 48 SERIES

Paso 1. Escribir el objeto: [7] [9] [ENTER]
Paso2. Escribir el nombre: ['] [α] [▭A] [α] [EEX] [ENTER]
Paso3. Almacenar el objeto: [STO]
Paso4. Ver el objeto: [VAR]

HP 49 SERIES

Paso 1. Escribir el objeto: [7] [9] [ENTER]
Paso2. Escribir el nombre: [↱] [EQW] [ALPHA] [FI] [ALPHA] [÷] [ENTER]
Paso3. Almacenar el objeto: [STO▶] [VAR]
Paso4. Ver el objeto: [STO▶]



La constante almacenada podrá ser utilizada en cualquier operación, únicamente pulsando la tecla correspondiente en el menú del usuario [VAR] [AZ]

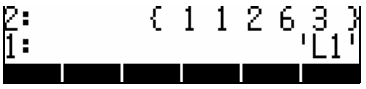
Ejemplo 05: *Almacenar la lista { 1 1 2 6 3 } con el nombre 'L1'*

HP 48 SERIES

Escribir el objeto: [←] [+] [1] [1] [2] [6] [3] [ENTER]
Escribir el nombre: ['] [α] [NXT] [1] [ENTER]
Almacenar el objeto: [STO]

HP 49 SERIES

Escribir el objeto: [←] [+] [1] [1] [2] [6] [3] [ENTER]
Escribir el nombre: [↱] [EQW] [ALPHA] [NXT] [1] [ENTER]
Almacenar el objeto: [STO▶]



Ejemplo 06: *Almacenar un programa (que sume 5+6), con el nombre de 'PRG1'*

HP 48 SERIES

Escribir el objeto: [←] [−] [5] [SPC] [6] [SPC] [+] [ENTER]
Escribir el nombre: ['] [α] [α] [◀] [▶] [MTH] [α] [1] [ENTER]
Almacenar el objeto: [STO]

HP 49 SERIES

Escribir el objeto: → + 5 SPC 6 SPC + ENTER
Escribir el nombre: → EQW ALPHA SYMB √x APPS / ENTER
Almacenar el objeto: STO▶

8.2. Recuperar un objeto (Editar el objeto).

Para recuperar un objeto del menú VAR (el menú VAR, contiene todos los objetos definidos por el usuario), consiste en presionar la tecla correspondiente al nombre del objeto del menú VAR
Ejemplo 07: *Recuperar el objeto almacenado en el ejemplo 4 (anterior)*

Buscar en el menú VAR el nombre del objeto A2 si no se encuentra en la pág.1 extendemos a la pág.2 con NXT hasta encontrar el nombre del objeto A2 una vez encontrado, pulse el mismo con la tecla correspondiente al nombre del objeto A2.

Otra forma de recuperar el objeto almacenado es, escribiendo en la pila el nombre del objeto.

HP 48 SERIES
HP 49 SERIES

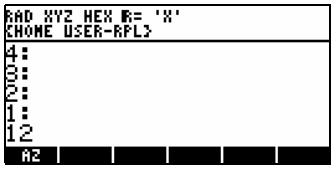
Escribir: α A α EEX ENTER
Escribir: ALPHA F1 ALPHA ÷ ENTER

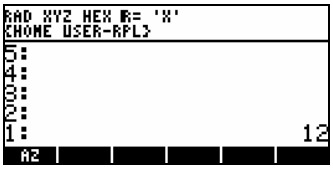
8.3. Guardar un objeto editado

Un atajo para colocar el objeto en la pila para edición solo presionamos → A2
Y una vez editado lo guardamos con el siguiente atajo ← A2

Ejemplo 08: *Editar el ejemplo 4 y asignarle (guardar) otro valor.*

Editar: Escribir otro valor: 1 2 ENTER
Guardar: ← A2





8.4. Borrar un objeto

Para borrar un objeto del menú VAR en el directorio actual, primero se ingresa el nombre del objeto en la pila, luego, pulse PURG
Ejemplo 09: *Borrar el objeto L1*

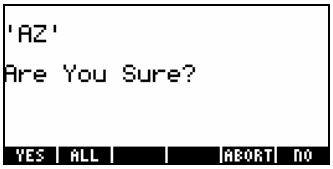
HP 48 SERIES
Escribir el nombre del objeto: ' α NXT 1 ENTER
Borrar el objeto: ← EEX
HP 49 SERIES
Escribir el nombre del objeto: → EQW ALPHA NXT / ENTER
Almacenar el objeto: TOOL PURGE

Otra forma de borrar un objeto es utilizando el localizador de variables (Administrador de objetos).

HP 48 SERIES
Para ingresar al localizador de variables pulse → MEMORY seleccione la variable o las variables que se desea borrar, luego pulse NXT PURGE

Ejemplo 10: *Borre AZ*
Escribir: → MEMORY NXT PURGE OK

HP 49 SERIES
← APPS OK NXT PURGE YES



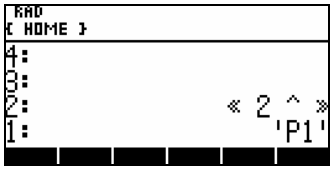
II. Fundamentos de programación.

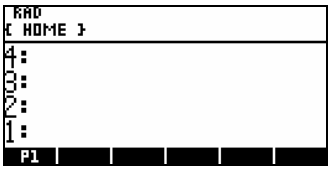
9. Qué es un programa.

Un programa es en general una secuencia de comandos, es decir todo el proceso para resolver un determinado problema, pasó a paso, lógicamente. Un programa para nuestra calculadora es nada más que un objeto definido por los delimitadores « », este contiene un conjunto de instrucciones, números u objetos.

Por ejemplo, si queremos realizar un programa para elevar al cuadrado un número puesto en el nivel-1 de la pila, nuestro programa será: « 2 ^ »

Ejemplo 11:
Escribir el programa: ↶ − 2 SPC y^x ENTER
Nombre del programa: ' α ◀ 1 ENTER
Almacenar el Prg.: STO





Para almacenar este programa en el directorio actual, en el menú VAR, se guardar como cualquier objeto.

9.2. Declaración de variables.

Una variable en nuestra calculadora es nada más que un objeto nominado por un nombre; que es el nombre de variable, estas pueden ser variables locales globales.

9.2.1. Variables Locales.

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

Ejemplo 12: *Realizar un programa que sume dos números puestos en la pila*

```
«
  → A B
  «
    A B +
  »
»
```

Guardamos el programa y para ejecutarlo necesitamos dos objetos tipo 0 puestos en la pila

En el ejemplo se muestra la secuencia para crear variables locales. El comando (→) es el encargado de recoger 2 valores de la pila y almacenarlos en las variables definidas; en este caso el nivel 2 se almacenara en A, el nivel 2 en B A continuación se ha iniciado otro programa o subprograma, esto se hará generalmente cuando se tengan programas complejos cuya secuencia de comandos sea mucho más extensa.

9.2.2. Variables Globales.

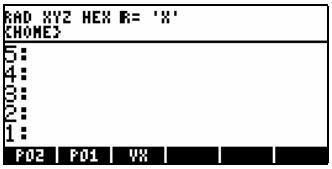
Las variables globales son todos los objetos que se tienen almacenados en la calculadora y las que se van a crear de manera permanente (por decirlo de alguna forma) manualmente ó por medio de los programas.

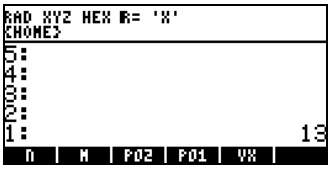
Su uso dentro de los programas representa una desventaja, ya que el trabajo con estas variables es un poco lento y para borrarlas necesitas hacer uso del comando PURGE.

En la mayoría de casos no se tiene necesidad de crear variables globales, ya que generalmente las variables se usan de manera temporal y es aquí donde entran las variables locales.

Ejemplo 13: *Realizar un programa que evalué* M^2+N^2

```
« 2 'M' STO
  3 'N' STO
  'M^2+N^2' EVAL
»
```



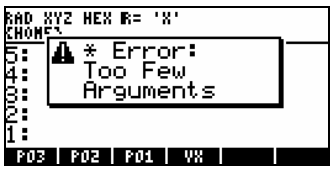


Al ejecutar el programa notaremos que han aparecido dos nuevas variables en el directorio actual. Si queremos que las variables no aparezcan tendríamos hacer uso de uso del comando PURGE

```
« 2 'M' STO
  3 'N' STO
  'M^2+N^2' EVAL
  'M' PURGE
  'N' PURGE
»
```

10. Entrada de datos.

Un programa requiere el ingreso de datos ya que sin esto siempre nos votara un mensaje: `Too Few Arguments`



La manera común es utilizar la pila para realizar este proceso, pero la calculadora dispone de comandos especiales para cumplir este objetivo, además permite darle una buena presentación a nuestros programas.

10.1. Comando INPUT.

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

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

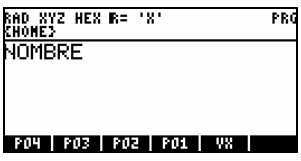
Donde:

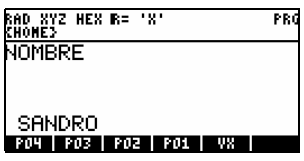
cadena: *sera el texto inicial, que aparezca en la línea de comandos*
fila y columna: *serán la ubicacion del cursor en la línea de comandos.*
modo(s): *será el modo por defecto con el que se presentara la línea de comandos.*

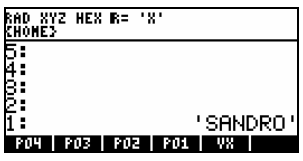
Modos:
`α` ,*para introducir una cadena de caracteres directamente.*
`ALG`, *para introducir objetos algebraicos.*
`V`, *para realizar una comprobación de la línea de comandos.*

Ejemplo 14:

```
« "NOMBRE"
{ " " α }
INPUT OBJ→
»
```

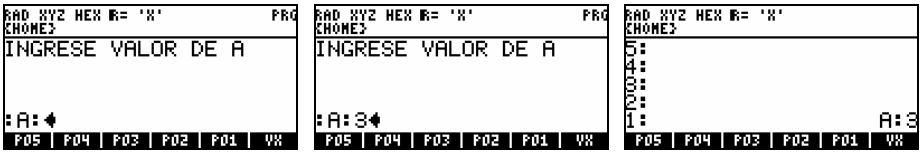






Ejemplo 15:

```
« "INGRESE VALOR DE A"
{ ":A:" (1 0) V }
INPUT OBJ→
»
```



10.2. Comando INFORM.

Comando utilizado principalmente para crear plantillas de entrada de datos, la sintaxis a seguir es:

```
{ ("Etiqueta" "Ayuda" Tipo1 Tipo2 Tipo3 ... TipoN ) ... }
```

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

Ejemplo 16:

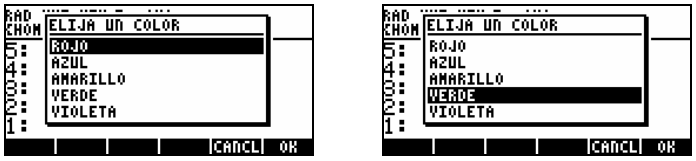
```
« "AREA DEL TRIANGULO"
{ "BASE" "ALTURA"}
{1 3} {} {}
INFORM
»
```

10.3. Comando CHOOSE.

Este comando muestra una lista de opciones en un pequeño cuadro por las cuales podemos desplazarnos y elegir la opción deseada. El siguiente ejemplo muestra como se puede programas y utilizar el comando choose con una ventan en miniatura desplegable.

Ejemplo 17:

```
« "ELIJA UN COLOR"
{"ROJO" "AZUL" "AMARILLO" "VERDE" "VIOLETA"} 1 CHOOSE »
```



11. Salida de datos.

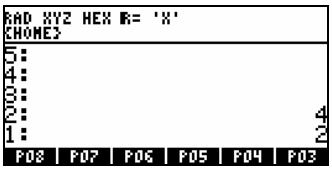
Luego de la ejecución de un programa ó durante ésta, debemos de presentar algunos resultados. Al igual que en el caso anterior (entrada de datos) la calculadora posee excelentes comandos para presentar datos.

11.1. MSGBOX.

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

Ejemplo 18:

```
« → R A
  «'π*R^2' →NUM 'A' STO
   "EL AREA DEL CIRCULO ES:"
   A →STR + MSGBOX
  »
»
```



11.2. DISP.

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

Ejemplo 19:

```
« → N
  « N DISP 0 FREEZE»
»
```

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

Argumentos	Zona congelada
0	Toda la pantalla
1	El área de Estado
2	La pila

3	El área de estado y la pila
4	Los menús
5	Los menús y el área de estado
6	La pila y los menús
7	Toda la pantalla

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

Ejemplo 20:

```
« → N
  « CLLCD N DISP 0 FREEZE»
»
```

Ejemplo 21:

```
« CLLCD "  FACULTAD
      DE INGENIERIA"
  2 DISP
  3 WAIT
»
```

El Comando `WAIT` es quien congela la pantalla temporalmente

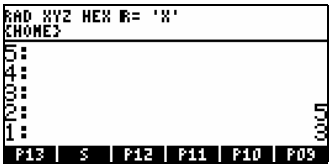
Ejemplo 22:

```
« → A B
  « A B +
    "A+B" →TAG
  »
»
```

El comando `→TAG` muestra en la pila como un objeto etiquetado

Ejemplo 23:

```
« → A B
  « A B + 'S' STO
   "SUMA DE A+B="
   S →STR + 500 .4 BEEP MSGBOX
  »
»
```



El comando BEEP permite resaltar una ejecución con indicadores audibles como sonidos. BEEP toma dos argumentos la frecuencia de tono y la duración del tono.

III. Estructuras de programación

12. Estructuras condicionales.

Las estructuras condicionales permiten que un programa tome una decisión basada en el resultado de una o más pruebas. Estas estructuras se encuentran disponibles en el menu [3:24] [IF] también en el MENU 24

12.1. IF…THEN…END.

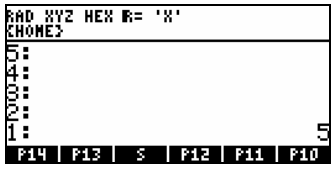
Podríamos traducirla como **SI...ENTONCES...FIN** de forma que explicaríamos su sintaxis:

«...IF *cláusula-prueba* THEN *cláusula-verdadera* END ...»

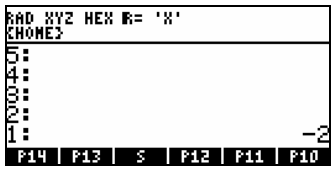
IF…THE…END ejecuta la secuencia de comandos en la acción solo si la acción es verdadera. IF inicia la condición, la cual deja un resultado de prueba en la pila. THEN retira el resultado de prueba de la pila. Si se cumple el valor de la acción, se ejecuta la acción; de lo contrario, la ejecución del programa se reanuda después de END.

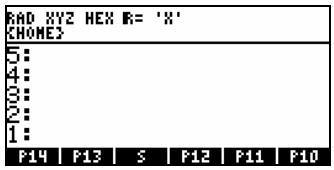
Ejemplo 24:

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



El programa tomará un valor (variable local N de la pila), comprueba que el número sea mayor que 0, si lo es entonces devolverá un mensaje, si no lo es no devolverá ningún valor.





12.2. IF…THEN…ELSE…END.

Traducida como SI...ENTONCES...SINO… FIN, la sintaxis para la estructura es:

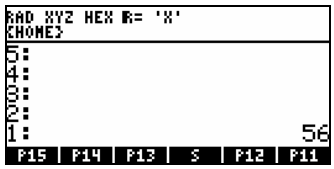
«...IF *cláusula-prueba*
THEN *cláusula-verdadera* ELSE *cláusula-falsa* END...»

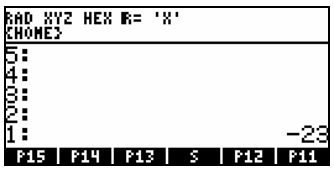
IF…THEN…ELSE…END ejecuta la secuencia de comandos en la acción si la condición es verdadera, o bien la secuencia de comandos acción diferente si la condición es falsa.

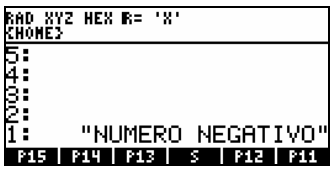
IF inicia la condición, la cual deja un resultado de prueba en la pila. THEN retira el resultado de la condición de la pila, si el valor es verdadero se ejecuta la acción verdadera; de lo contrario se ejecuta la acción diferente.

Ejemplo 25:

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







El programa tomará un valor (variable local N de la pila), comprueba que el número sea mayor que 0, en cualquier caso devolverá el mensaje de acuerdo al número introducido.

12.3. CASE…THEN…END.

Para la estructura **CASE**…**THE**…**END**
EN CASO DE…**ENTONCES**…**FIN** su sintaxis es la siguiente

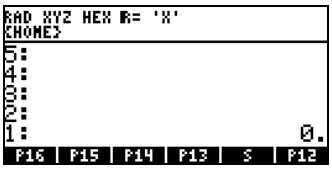
```
«... CASE
      cláusula-prueba-1 THEN cláusula-verdadera-1 END
      cláusula-prueba-2 THEN cláusula-verdadera-2 END
  .
  .
  .
      cláusula-prueba-N THEN cláusula-verdadera-N END
    cláusula-por defecto (opcional END
                              ...»
```

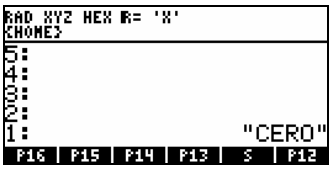
Al ejecutarse CASE, se evalúa la *condición 1*. Si la prueba es verdadera, se ejecuta la *acción 1*, y la ejecución salta a END. Si la *condición 1* es falsa la ejecución pasa a la *condición 2…*

La ejecución dentro de la estructura CASE continúa hasta que se ejecuta una *condición* verdadera o hasta que todas las condiciones dan falso como resultado. Se incluye una *acción por defecto* si todas las condiciones dan falso como resultado, se ejecuta la *acción por defecto.*

Ejemplo 26:

```
« → N
  « CASE N 0 >
    THEN "NUMERO POSITIVO" END
    N 0 <
    THEN "NUMERO NEGATIVO" END
    "C E R O"
    END
  »
»
```





CASE evalúa que N es mayor a cero, luego si N es menor a cero y si no, se asume esta como cero, en cualquier caso se devolverá el mensaje de acuerdo al número introducido.

12.4. IFERR…THEN…END.

Es una estructura de detección de errores en la ejecución de programas. Las estructuras de detección de errores permiten que lo programas detecten (o interpreten) las situaciones de error que de lo contrario, provocarían la suspensión de la ejecución del programa.

IFERR responde a la necesidad de poder controlar errores en la ejecución de programas. Esta estructura tiene gran semejanza con el IF, su traducción es la siguiente:

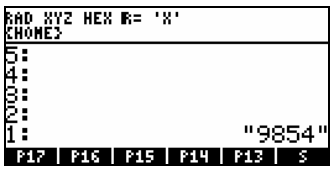
SI se produce un ERROR…**ENTONCES**…**FIN**

La sintaxis para esta estructura es la siguiente:

«...IFERR *cláusula-detección* THEN *cláusula-error* END...»

Ejemplo 27:

```
« IFERR
  IF 'A*3'
  THEN
  CLLCD "Depende de A"
  1 DISP 7 FREEZE
  END
  THEN
  CLLCD "Has hecho algo mal"
  1 DISP 7 FREEZE
  END
»
```



Si la variable 'A' no existe o contiene un objeto que produce un error al ser evaluado por THEN el programa no se interrumpe sino que salta a la cláusula THEN de IFERR y se ejecuta esta: CLLCD "Has hecho algo mal" 1 DISP 7 FREEZE . En caso de que no se produzca ningún error esta cláusula no se ejecuta.

12.5. IFERR…THEN…ELSE…END.

Sería la estructura anterior a la que se le dota de ELSE. Su traducción es la siguiente:
SI se produce un ERROR…**ENTONCES**…**SINO**…**FIN**
La sintaxis para esta estructura es la siguiente:

```
«...IFERR cláusula-detección
THEN cláusula-error ELSE cláusula-normal END...»
```

Ejemplo 28:

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

El programa del ejemplo anterior muestra los directorios en la ruta actual, pero ocurrirá un error si no existen directorios, en tal caso borramos los argumentos y no se muestra nada, pero si existen mostrará los directorios en un cuadro de elección en donde si seleccionas alguno te cambiarás a ese directorio

12.5.1. Comando IFT.

El comando IFT es una versión compacta de IF...THEN...END

Ejemplo 28:

```
« → N
  « N 0 >
    "NUMERO POSITIVO"
    IFT
  »
»
```

13. Estructuras Iterativas.

Las estructuras iterativas, o bucles, permiten que un programa ejecute una secuencia de comandos varias veces. Para especificar por adelantado cuántas veces ha de repetirse el bucle utilice un bucle definido, para utilizar una prueba que determine si hay que repetir o no el bucle utilice un bucle indefinido.

13.1. Estructuras definidas.

Los bucles definidos llevan asociados una variable contador en la que se especifica el número de veces que ha de repetirse el código. Hay dos tipos de bucles definidos, cada uno de ellos tiene dos variantes dependiendo de si el contador se incrementa de uno en uno o de forma definida por el programador.

13.1.1. START…NEXT.

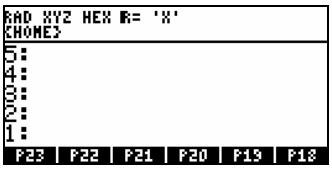
La estructura se traduce como INICIO...SIGUIENTE Su sintaxis:

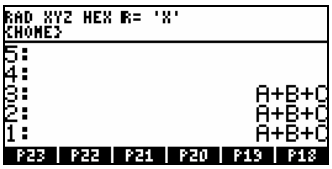
«. . . *inicio final* START *cláusula-bucle* NEXT. . . »

START toma dos números (inicio y final) de la pila y los almacena como los valores iniciales y final de un contador de bucle. A continuación, se ejecuta la acción. NEXT incrementa el contador por 1 y comprueba si su valor es menor que o igual a final. Si lo es, se vuelve a ejecutar la acción; de lo contrario, se reanuda la ejecución con NEXT, que viene a continuación.

Ejemplo 29:

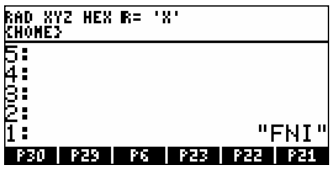
```
« 1 3
  START 'A+B+C'
  NEXT
  »
»
```





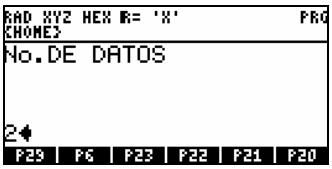
Ejemplo 30: *Programa que suma objetos de tipo 0*

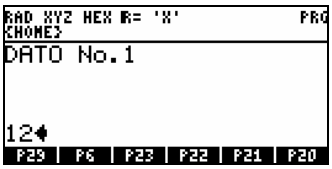
```
« "F" "N" "I"
  1 2
  START +
  NEXT
»
```

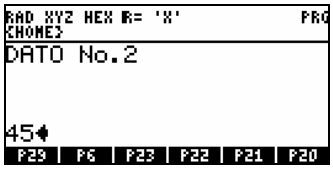


Ejemplo 31: *Programa para construir una lista de N elementos.*

```
« 0 → X
  « "No.DE DATOS" ""
    INPUT OBJ→ 1 SWAP
    START "DATO No." 'X'
    INCR + "" INPUT OBJ→
    NEXT DEPTH →LIST
  »
»
```







El comando `INCR` (incremento) toma un nombre de variable local o global como argumentos, el contador devuelve el nuevo valor de la variable, incrementa en 1 el valor almacenado en la variable.

13.1.2. STAR…STEP.

La estructura START...STEP se podría traducir como:
INICIO... INCREMENTO, su sintaxis sería:

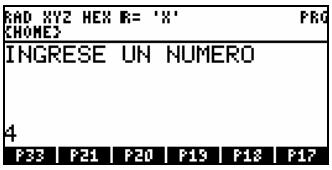
« . . . inicio final START *cláusula-bucle*
incremento STEP *. . . »*

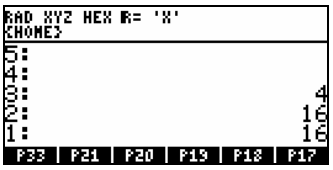
Esta estructura sólo se diferencia de la anterior en que se puede especificar un incremento distinto de 1, que era el único que podíamos tener con la anterior.

El incremento puede ser positivo o negativo (nunca cero ya que el bucle sería infinito), si usamos un algebraico o un nombre de variable como argumento de STEP se evalúa y su resultado es con el que STEP incrementa el contador. En caso de que el incremento sea negativo 'inicio' debe ser mayor que 'fin' deteniéndose el bucle cuando el contador es menor que el valor de 'fin'

Ejemplo 32: *programa que toma un número N y calcula el cuadrado de ese numero dos veces*

```
« "INGRESE UN NUMERO"
  "" INPUT OBJ→ DUP → N
  « N 1
    START N SQ -2
    STEP
  »
»
```





13.1.3. FOR…NEXT, FOR…STEP.

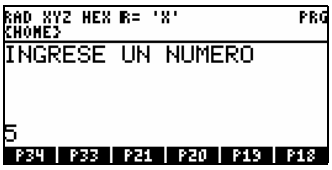
Estas estructuras son idénticas a las anteriores, la diferencia que estas poseen un contador. En realidad son las estructuras más utilizadas ya que poseen la mayoría de características para poder programar, la sintaxis a seguir es:

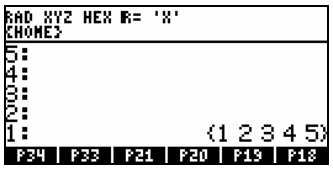
«. . . *inicio final* FOR *contador cláusula-bucle* NEXT. . . »

FOR inicia la iteración, se declara una variable contador y **NEXT r**ealiza la comprobación y continua ó finaliza la iteración, por defecto el contador para las iteraciones se incrementa en 1, si deseas decrementos u otros incrementos debes de utilizar **STEP**, en cuyo caso debemos colocar los incrementos antes de esta instrucción.

Ejemplo 33: *Programa que genera una lista dado N como No. de elementos de la lista.*

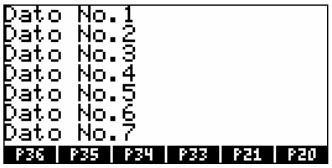
```
« "INGRESE UN NUMERO"
  "" INPUT OBJ→ → N
  « {} 1 N
    FOR i i +
    NEXT
  »
»
```





Ejemplo 34: *programa que imprime los 5 primeros números*

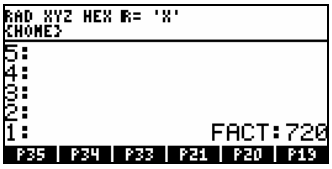
```
« CLLCD
  1 7
  FOR i
  "Dato No."
  i + i DISP
  NEXT 7 FREEZE
»
```



Ejemplo 35: *programa para hallar el factorial de un número*

```
« "FACTORIAL DE N"
  "" INPUT OBJ→ → N
```

```
  « IF 'N==0' THEN 1
    ELSE 1
      1 N
      FOR i
      i *
      NEXT
    END
"FACT" →TAG
  »
»
```



13.2. Estructuras indefinidas.

13.2.1. DO…UNTIL…END.

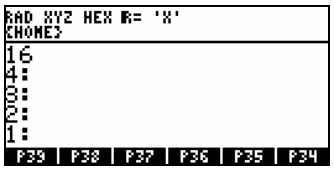
Esta es una estructura iterativa, la diferencia con las anteriores es que requiere de una condición para finalizar; la sintaxis a seguir es:

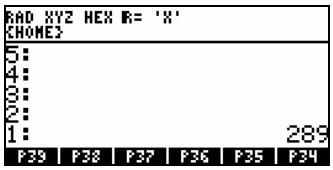
«... DO *cláusula-bucle* UNTIL *cláusula-prueba* END...»

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

Ejemplo 36:

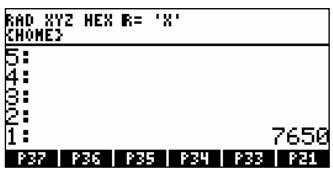
```
« 0
  DO DUP 1 DISP 1 +
  UNTIL KEY
  END DROP
»
```





Ejemplo 37:

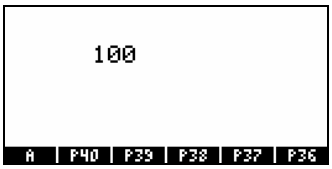
```
« 0 100 → SUM N
  « DO 'SUM'
    N STO+ 2
    'N' STO+
    UNTIL N 200 >
    END
    SUM
  »
```

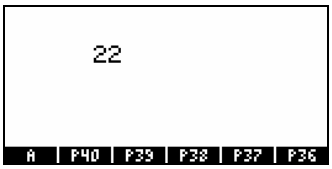


```
»
```

Ejemplo 38: *Programa muestra el decremento de 100 a 0*

```
« 100 'A' STO CLLCD
  "      " A + 3 DISP .25 WAIT
  DO CLLCD
  "      " 'A' DECR +
  3 DISP .25 WAIT
  UNTIL 'A==0'
  EN 'A' PURGE
»
```





13.2.2. WHILE…REPEAT…END.

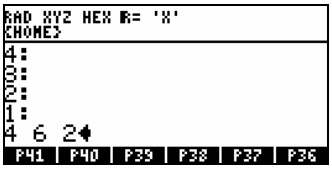
Esta es otra estructura iterativa, la diferencia con la anterior es que requiere de una condición para finalizar y no se ejecuta nunca si la condición de prueba es falsa.; la sintaxis a seguir es:

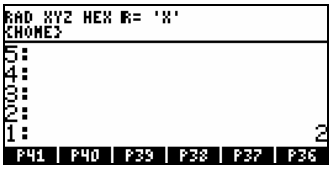
«... WHILE *cláusula-prueba* REPEAT *cláusula-buble* END...»

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

Ejemplo 39: *Programa para calcular el MCD de a y b*

```
« → A B C
 « WHILE A B MOD
   REPEAT
    A B MOD 'S' STO
    B 'A' STO
    C 'B' STO
   END B
 »
»
```

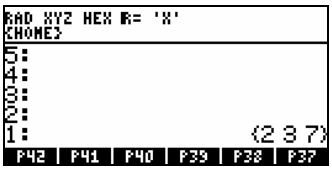




Ejemplo 40: *Programa que descompone en factores primos un números*

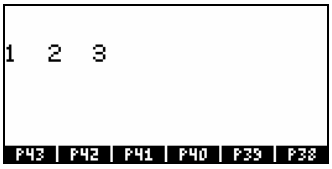
```
« "DESCOMPONER
EN FACTORES PRIMOS
El numero N="
"" INPUT OBJ→ 2 → A B
  « {}
    WHILE A 1 >
```

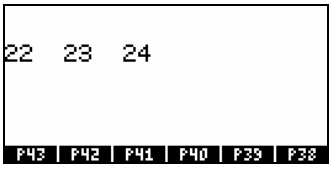
```
    REPEAT
      IF A B MOD 0 ==
      THEN B + A B /
        'A' STO
      ELSE B 1 +
        'B' STO
      END
    END
  »
»
```



Ejemplo 41: *Programa que muestra números de 1 a 50*

```
« → N
  «
   WHILE N 50 <
   REPEAT CLLCD ""
    'N' INCR + "  " + N 1
    + + "  " + N 2 + + 3
    DISP -1 WAIT DROP N 2
    + 'N' STO
   END
  »
»
```





IV. Introducción a System-RPL

14. RPL del Usuario y RPL del Sistema.

No hay ninguna diferencia fundamental entre el lenguaje de programación de la HP, "RPL de usuario" y "RPL del sistema" en el que está implementada la funcionalidad de la HP. Los programas en el lenguaje de usuario son ejecutados por el mismo bucle interno del intérprete que los programas del sistema, con la misma pila de retornos. La pila de datos que se muestra en la HP es la misma que la usada por los programas del sistema. La distinción entre RPL de usuario y RPL del sistema es solo una cuestión de alcance: el RPL de usuario es un subconjunto del RPL del sistema. El RPL de usuario no proporciona acceso directo a todos los tipos de objetos de la clase de datos disponibles; el uso de los procedimientos incorporados está limitado a aquellos proporcionados como comandos. Escribir programas en RPL del sistema no es diferente en principio que en RPL de usuario; la diferencia está en la sintaxis y en el alcance del compilador. Para el RPL de usuario, el compilador es el ENTER de la línea de comandos, el compilador para RPL del sistema es un programa (Compilador; p.ej el JAZZ) que analiza y traduce el texto del código fuente al lenguaje de la HP (Lenguaje ensamblador).

14.1. Programación en Sys-RPL

Programar en System resulta más peligroso pero a la vez más rápido, mas peligroso por que si se introduce un dato incorrecto puede ocasionar la perdida de la memoria ya que este lenguaje no verifica los argumentos. Por ejemplo si en UserRPL nosotros ejecutamos "+" y si no hay ningún argumento en la pila este mandara un mensaje de error, pero en SysRPL si ejecutamos el mismo programa y si no existe ningún argumento en la pila este mandara un `Try To Recovery Memory?` y en algunos casos ya no se podrá recuperar la memoria ocasionando la perdida de lo que tengamos grabado en el puerto 0 de la calculadora.

Para hacer un programa en SysRPL la sintaxis cambia es decir para iniciar un programa en User nosotros colocamos (`«...»`) contenido del programa en user en cambio para hacer un programa en System nosotros tenemos que colocar lo siguiente:

```
::
  Contenido del programa en system
;
```

14.2. Ventajas de programar en System-Rpl frente al User-Rpl.

Obviamente por que es más rápido y además con este lenguaje de más bajo nivel se puede tener acceso a muchos tipos de objetos que con el UserRPL seria imposible. La mayor razón de la diferencia en la velocidad es el hecho de que los comandos en UserRPL tienen incorporada revisión de argumentos y errores. En SysRPL, el

programador es responsable de todas las revisiones de errores para evitar el borrado de la memoria.

14.3. ¿Que se necesita para programar en System-Rpl?

La programación en System se puede hacer a través de programas especiales desde el computador, o desde la calculadora, Si vamos a programar en System RPL desde la calculadora necesitamos un programa compilador (un programa traductor que convierte los códigos del programa a un código que la HP lo entienda). Entonces para programar desde la calculadora se puede utilizar el poderoso y famoso compilador Jazz 6.8, una herramienta de programación muy completa. Existe también una variante, el Jazz Light 6.7, el cual ocupa menos espacio en la calculadora, pero no trae el DEBUGGER para probar los programas paso a paso. Adicionalmente se debe tener instalada las librerías de etiquetas o mnemónicos HPTABS y la librería de fuentes universales UFL. Entonces instalamos las siguientes librerías:
La librería JAZZ 6.8, 992
La librería HPTABS, 993
Librería de fuentes universales UFL es la 257

Una vez instalado estas herramientas en la HP o Emulador ya podemos empezar a hacer nuestros programas en sys-Rpl.

Comandos de JAZZ

ASS *Compila el código fuente de un programa.*
DIS *Descompila un ejecutable para convertirlo en un equivalente de su código fuente. También descompila cualquier tipo objeto.*
SDB *Ejecuta paso a paso un programa.*
ED *Editor de programas en código fuente o ejecutables.*

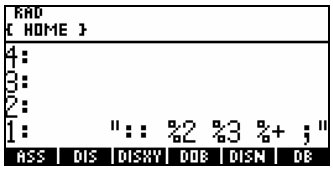
Ejemplo 01: *Programa que suma dos números*

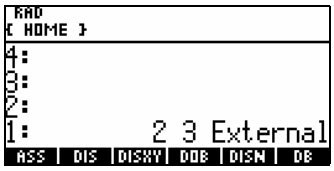
Sys-RPL

```
::
  %2 %3 %+
;
```

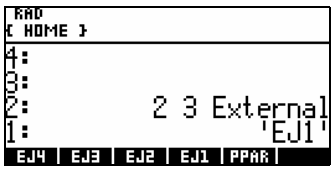
User-RPL

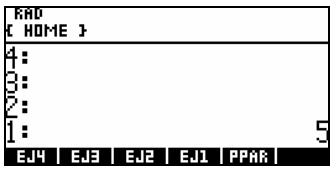
```
«
  2 3 +
»
```





Paso 1: Colocar el programa en la pila como una cadena.
Paso 2: ▶ 2 





Paso 3: Guardar el programa `'EJ1'` STO

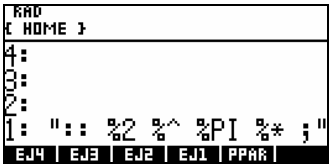
Ejercicio 02: *Programa que calcula el área de un círculo.* `πR^2`

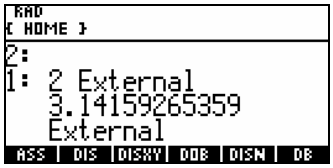
Sys-RPL

```
::
  %2 %^ %PI %*
;
```

User-RPL

```
«
  2 ^ π * →NUM
»
```





Guardamos y ejecutamos el programa, necesitando un valor en la pila para R

14.4. ¿Qué es un Emulador?

Un emulador es un software que permite a un equipo, en nuestro caso una PC, ejecutar las acciones de otro, nuestra calculadora. Se trata por tanto de un programa que permite ejecutar en nuestra PC los comandos y programas de nuestra calculadora.

Primera ejecución del Emulador (EMU48)

Para ejecutar Emu48 debes hacer doble clic en el archivo Emu48.exe o bien si hiciste un acceso directo puedes hacer clic sobre él. Lo primero que verás será lo siguiente: