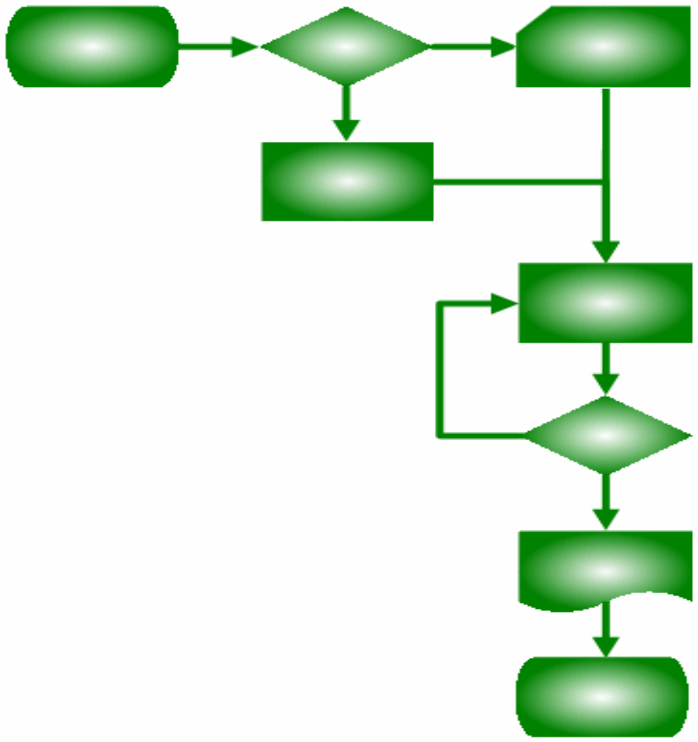
FUNDAMENTOS DE USER RPL

Programación de HP48 para Principiantes

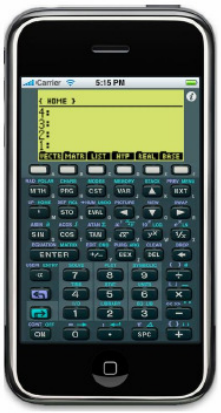


POR
MARCOS A. NAVARRO

FUNDAMENTOS DE USER RPL

Edición Revisada y Corregida

Enero 4, 2011



CONTENIDO

INTRODUCCIÓN

Las calculadoras HP han sido las preferidas de muchos estudiantes, técnicos y profesionales alrededor del mundo. Pueden ser programadas en varios lenguajes y en el presente manual se da a conocer el más amigable de todos ellos: ¡El Lenguaje User Rpl!

Este manual aspira servir de apoyo a aquellas personas que se inician en el mundo de la programación en User Rpl. Aquí son expuestos temas fundamentales de dicho lenguaje de una forma gradual y mediante el empleo de diversos elementos didácticos.

Como forma de contribuir a la adecuada comprensión de los temas se presentan variados ejemplos y ejercicios para ser analizados y resueltos por el lector. También se hace uso de diagramas de flujo, y muchas informaciones se presentan en cuadros y tablas. Tratando de hacer más ameno el material, se presentan algunos textos con informaciones que pudieran resultar interesantes o curiosas para el lector.

Durante el estudio de este manual, se debe tener a mano la ***Guía de Usuario*** de la Serie HP48. Además puede ser útil un emulador de HP48. Los foros de Internet son un gran apoyo que debe aprovecharse, en estos participan estudiantes y profesionales de Ciencias e Ingenierías de todo el mundo.

Quiero agradecer a todas las personas que han leído el manual, a quienes les dedico esta edición.

Marcos A. Navarro
ing.mnavarro@hotmail.com

1.1 Guía de Usuario en español: http://h10032.www1.hp.com/ctg/Manual/bpia5247.pdf

2.1 Emulador para PC (Windows): http://hp.giesselink.com/emu48.htm
2.2 Emulador para Palm (Palm OS 5): http://power48.softonic.com/palm
2.3 Emulador para iPhone: http://www.mksg.de/m48/m48.html

3.1 Foro en español: http://www.adictoshp.org/forum
3.2 Foro en Ingles: http://groups.google.com/group/comp.sys.hp48

4.1 Actualizaciones de este manual: http://www.hpcalc.org

CAPÍTULO I. ¡Comencemos Ya!

1.1 Los lenguajes de la HP48.

Los programadores crean Software utilizando algún lenguaje de programación.

La HP48 puede ser programada en varios lenguajes, siendo el User Rpl el más amigable de todos.

El User Rpl es un leguaje muy seguro. Impide la presencia de errores que, en caso de ocurrir, pueden ocasionar fallas en el sistema, pérdida de información y otras situaciones no deseadas. Estas protecciones contra errores, y otros gastos que tienen lugar cuando se usa este lenguaje, son la causa de la diferencia de velocidad entre el User Rpl y lenguajes más potentes, como son: El System Rpl y el Saturno Assembler.

Se recomienda aprender User Rpl antes de estudiar System Rpl. Aunque, lo cierto es que este último es un subconjunto del primero.

El lenguaje Saturno Assembler, por su parte, crea programas en código máquina (ML) los cuales se ejecutan directamente sobre el cerebro de la calculadora. Es el lenguaje más poderoso de todos y requiere buen conocimiento del procesador por lo que es el lenguaje más complejo. Se dice que el Saturno Assembler es un lenguaje de bajo nivel, mientras el User Rpl es de alto nivel.

1.2 Los Objetos en los Programas.

Los programas, son una sucesión de objetos que la calculadora ejecuta para realizar una tarea específica.

Los objetos, son elementos básicos utilizados por la HP48. Por ejemplo, un número real o complejo, una matriz, una lista, un objeto gráfico, un programa y una librería son, cada uno de ellos, un objeto.

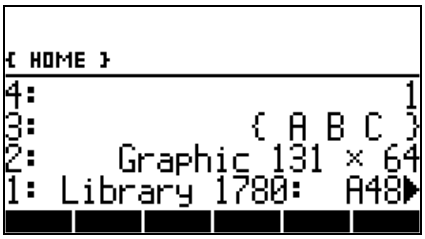


figura 1.1

1.3 Existe un tipo especial de objetos llamados comandos.

Otro tipo de objetos son los llamados comandos. Estructuralmente, todos los comandos son programas almacenados en una librería y que poseen un nombre por el cual pueden ser llamados.

Como ejemplo podemos citar el comando **MSGBOX**, el cual toma un texto de la pila y crea un recuadro de mensaje.

Para ver cómo es esto, encienda su HP48 y escriba un texto cualquiera entre comillas dobles (“TEXTO”). Luego presione ENTER para enviarlo a la pila.

Con el texto en el nivel **1:** de la pila, escriba el comando **MSGBOX** y presione ENTER (ver figura 1.2).

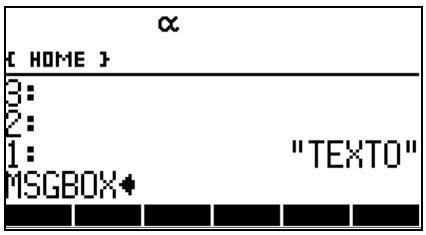


figura 1.2

El comando **MSGBOX** creara un recuadro de mensaje con el texto previamente suministrado.

figura 1.3

Otro comandos que podemos citar es el comando **BEEP**, el cual produce un pitido, dada la frecuencia (en hertz), en el nivel **2: ;** y el tiempo de duración (en segundos), en el nivel **1:**

El ejemplo de la figura siguiente produce un pitido de 1500 hertz, durante 1 segundo:

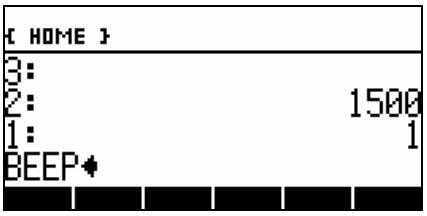


figura 1.4

Es importante aclarar que no todos los comandos requieren argumentos en la pila. El comando **RAD**, por ejemplo, no requiere nada en la pila y lo que hace es activar el modo de ángulos en radianes.

En el **Apéndice G** de la *Guía de Usuario* de la HP48 se encuentra la lista de comandos.

Usando el Apéndice G de la *Guía de Usuario* , investigue qué hacen los siguientes comandos:

0	**OFF**	5	**SQ**	10	**SIN**
1	**MEM**	6	**FP**	11	**STO**
2	**DUP**	7	**+**	12	**VARS**
3	**SWAP**	8	**ABS**	13	**PGDIR**
4	**DROP**	9	**DEG**	14	**TRN**

1.4 Hagamos nuestros primeros programas.

Tome su HP48, presione la tecla ← y luego la tecla −.

Deben aparecer los símbolos << y >>. Estos son los delimitadores de programa. Es dentro de estos símbolos que se debe escribir los comandos y otros objetos que formarán parte de nuestros programas.

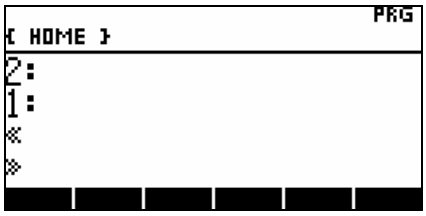


figura 1.5

Ahora, dentro de los delimitadores, escriba un texto cualquiera (entre comillas dobles) seguido por el comando **MSGBOX.**

figura 1.6

Obsérvese que debe haber un espacio entre el texto y el comando.

Presione **ENTER** para enviarlo a la pila. Y, finalmente, presione **EVAL.**

Al evaluar el programa se obtiene el mismo resultado que si colocamos el texto en la pila y ejecutáramos el comando MSGBOX.

Ya hemos escrito nuestro primer programa, el cual, aunque muy simple, está bien para ser el primero.

Hagamos ahora otro programa que presente el texto:

"BIENVENIDOS AL USER RPL"

y luego produzca un pitido de 1500 hertz, durante 1 segundo:

<< "BIENVENIDOS AL USER RPL"

MSGBOX 1500 1 BEEP >>

¿Cómo sería, si quisiéramos que el pitido se produjera antes del mensaje? En este caso escribimos:

<< 1500 1 BEEP "BIENVENIDOS AL USER RPL" MSGBOX >>

También, podemos introducir el comando **OFF** para hacer que la calculadora se apague después de presentar el mensaje:

<< 1500 1 BEEP "BIENVENIDOS AL USER RPL" MSGBOX OFF >>

Como se ha podido apreciar, las instrucciones se evalúan de izquierda a derecha: primero el pitido, luego el mensaje y al final el **OFF.**

1.5 El User Rpl es Case Sencitive.

En el User Rpl, no es lo mismo escribir OFF que Off, ni que oFF. La HP48 distingue entre mayúsculas y minúsculas, por esto se dice que es case sencitive (sensible a las mayúsculas y minúsculas).

1.6 Guarde sus programas como variables globales.

Después de escribir un programa, debemos guardarlo como variable. De esta manera podremos ejecutarlo todas las veces que queramos, utilizando su nombre o la tecla de VAR que le corresponda.

Podemos usar cualquier nombre para nuestros programas. Eso si, siempre que no sea el nombre de un comando u otros nombres prohibidos.

Los nombres pueden contener hasta 127 caracteres.

Si el lector tiene problema con la manipulación de variables, es urgente que repase la *Guía de Usuario* antes de continuar con sus estudios de User Rpl. De todas formas explicare algunos detalles básicos:

1) Para guardar cualquier objeto solo tenemos que colocarlo en el nivel 1: , luego escribes su nombre (sin comillas) y presionas la tecla STO.

2) Para recuperar el contenido de una variable a la pila, pulse  y a continuación la tecla del menú **VAR** de la variable.

3) Para recuperar el nombre de una variable a la pila, pulse la tecla  y a continuación la tecla del menú **VAR** de la variable.

4) Para borrar una variable solo tenemos que recuperar su nombre a la pila y presionar  y luego **PURG.**

1.7 Las modificaciones de programas son frecuentes.

Luego de escribir un programa, es común que tengamos que hacerle pocas o muchas modificaciones para que llegue a funcionar en forma satisfactoria.

Si el programa está guardado en VAR, se suele recuperar su contenido en la pila y luego presionar  para comenzar con la edición. Terminadas las modificaciones, se presiona **ENTER** para enviar el programa a la pila, y luego  y la tecla de la variable para actualizarla.

1.8 La pila de datos.

La HP48 se utiliza mediante la introducción de objetos en la pila. y la ejecución de los comandos que operan sobre los datos.

Aunque solo son mostrados cuatro niveles de pila, el número de niveles solo está limitado por la cantidad de memoria disponible.

Cuando se introduce un nuevo dato en la pila, el dato que estaba en el nivel **1:** pasa al nivel **2:**; el que estaba en el **2:** pasa al nivel **3:**, y así sucesivamente.

La pila es esencialmente LIFO (last in, firt out: Último que entra, primero que sale), pero podemos cambiar el orden de sus datos, utilizando los llamados "Comando de Pila". Podemos, por ejemplo, invertir el orden de los niveles **1:** y **2:**, utilizando el comando **SWAP** ; o mover el dato del nivel **3:** hacia el nivel **1:**, utilizando **ROT.** Otro comando de pila es **DROP,** el cual elimina el objeto del nivel **1:.**

Si tenemos por ejemplo:

Nivel 2: 5
Nivel 1: 3

al ejecutar **<< SWAP>>** , resultara:
Nivel 2: 3
Nivel 1: 5

Lo que hace el SWAP es mover el objeto del nivel 2: hacia el nivel 1:, y el del 1: hacia el 2:.

y si tenemos:
Nivel 3: 1
Nivel 2: 7
Nivel 1: 5

al ejecutar **<< ROT >>,** resultara**:**
Nivel 3: 7
Nivel 2: 5
Nivel 1: 1

Fíjese que ha ocurrido una especie de rotación: el objeto del nivel 3: se ha movido al 1:, y los demás han subido un nivel.

Y si a este último resultado le aplicamos **<< DROP>>,** obtendremos:
Nivel 1:
Nivel 3: 7
Nivel 2: 5

El DROP ha eliminado el objeto del nivel 1: y los demás han descendido un nivel.

Otro comando de pila es **DUP,** el cual duplica el objeto del nivel **1:.** Si tenemos por ejemplo:
Nivel 2:
Nivel 1: 9

Al aplicar **DUP** obtendremos**:**
Nivel 2: 9
Nivel 1: 9

Más adelante, estudiaremos otro comando de pila. Por ahora nos conformaremos con los anteriores.

Escribamos un programa que tome tres objetos de la pila y elimine el del nivel 3**:.**

<< ROT DROP >>

Escribamos otro, que elimine el objeto del nivel 2:

<< SWAP DROP>>

Ejercicio: Escriba un programa que elimine los datos en 1: y 3:

1.9 ¿Qué tal un poco de Matemáticas? Primero hagamos un programa muy simple que tome un número de la pila, le sume 3 y luego divida por 2. El código será:

<< 3 + 2 / >>

Guárdelo en una variable y pruébelo.

Veamos en detalle cómo opera este pequeño programa. Para esto llamaremos **número dado** al número que se pone en la pila antes de ejecutar el programa**.**

El primer evento que ocurre es el envío de un 3 a la pila. Entonces tenemos:

2: número dado
1: 3

luego, el comando **+** suma el 3 con el número dado:

2:
1: (número dado + 3)

a continuación, se envía el número 2 a la pila:

2: (número dado + 3)
1: 2

y finalmente ejecuta el comando **/**

2:
1: (número dado + 3)/ 2

Claro que todo lo anterior ocurre a una velocidad tremenda. Pero, le adelanto que existe una forma de ver, en la calculadora, el funcionamiento paso a paso de los programas. Estos asuntos los trataremos detenidamente más adelante.

En el programa anterior, lo último que se hacía era dividir por 2. Pero, ¿cómo sería el código si lo que queremos es dividir 2 por el número que esté en la pila? En este caso bastaría:

<< 2 SWAP / >>

Escribamos otro, que tome dos números de la pila, los reste y el resultado lo eleve al cuadrado.

Para esto se puede utilizar **^**:

<< - 2 ^ >>

Otra forma sería: **<< - DUP * >>**

Los comandos de pila pueden llegar a ser muy útiles a veces.

El programa anterior también pudo escribirse como:

<< - SQ >>

El comando **SQ** produce el mismo resultado que **2 ^** y ocupa menos memoria.

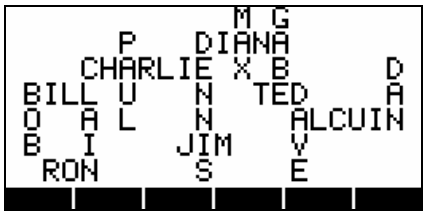
1.10 Una sugerencia para el lector.

Este momento es oportuno para que el lector haga una revisión del capítulo 12 de la *Guía de Usuario* de la HP48. En ese capítulo se describen los comandos para Trabajar tanto con números Reales como con Complejos.

No se preocupe por memorizar los comandos, solo trate de comprender cómo funcionan y si se le ocurre alguna idea escriba algunos programas.

Curiosidades

Existe un comando que resulta particularmente curioso. Este es el comando RULES, el cual no aparece en el apéndice G y lo que hace es mostrar un crucigrama de varios nombres. Pero inmediatamente surge una pregunta: ¿de quiénes son esos nombres?



ALCUIN fue el nombre código de la HP48G durante su desarrollo. Los otros nombre corresponden a los expertos que formaron parte del equipo de personas que la desarrollo:

BILL Wickes (proceso de lista; "Padre del RPL")
CHARLIE Patton (Sistema operativo RPL)
TED Beers (Pila interactiva; Sistema manipulación de programas; Manejo del despliegue; Formas de entrada)
DIANA Byrne (Manager del proyecto; Trazados; Gráficos)
GABE Eisenstein (EquationWriter)
BOB Worsley (I/O)
PAUL McClellan (Manejo de unidades; Matemáticas)
CLAIN Anderson (Manager del producto; Marketing)
DENNIS York (Manager)
JIM Donnelly (EQ LIB; proceso de listas; diseñador de apoyo)
MAX Jones (sistema de menú; edición)
DAVE Arnett (hardware)
DAN Coffin (manuales)
RON Brooks (mercadeo)

Inf. extraída de GD9/Postings/Team48g.doc

Ejercicios: Dados tres números x, y, z, en la pila:

3: z
2: y
1: x

(a) Escriba programas que calculen:

1) (x+y) / z
2) y/x
3) y+z
4) (x-z) *y
5) (x^y)^z
6) [(x^2)-(Y^2)]/z
7) (x*y)+(x*z)
8) (2*x+3*y)/z
9) x^z
10) (x^2)-2xy+(y^2)

nota: en este último se puede ahorrar memoria si programamos para: (x-y)^2

(b) Usando los comandos: trigonométricos SIN, COS, TAN, escriba programas para:

1) sin(x)+cos(y)
2) [sin(x)]^2 + [cos(x)]^2
3) 1+[tan(x)]^2

(c) El comando LN sirve para calcula el Logaritmo Neperiano. Utilícelo para:

1) 2* ln(x) ln(x+y)

(d) Escriba programas para calcular el volumen de:

1) un cilindro.
2) una esfera.
3) un cono.
4) una pirámide.

(e) Utilice el comando √ para programas que calculen:

1) √(x+y)
2) 1/√x
3) √[(x^2)+(y^2)]

(f) Utilizando el apéndice G de la *Guía de Usuario*, investigue para qué sirven los siguientes comandos:

1) BYTES
2) CLEAR
3) RCL
4) DROP2
5) DUP2

CAPÍTULO II. Variables y Subrutinas

2.1 Usando variables en nuestros programas.

2.1.1 Variables Globales.

En ejemplos anteriores, guardamos nuestros programas como variables globales utilizando la tecla **STO.**

También podemos crear variables globales desde nuestros programas. Es posible crear programas que tomen datos de la pila y los almacene como variables. Para esto, usaremos el comando **STO.** Este comando requiere un nombre en **1:** y el objeto a almacenar, en **2:.**

2: Objeto a Almacenar
1: 'Nombre'

Hagamos un programa que tome dos números de la pila y nos devuelva su suma, resta, producto y cociente. Para esto guardaremos el número del nivel **2:** con el nombre **'N2'** y al del nivel **1:** con el nombre **'N1'.**

```
<<
N1 STO N2 STO
N2 N1 +
N2 N1 −
N2 N1 *
N2 N1 /
{ N1 N2}  PURGE >>
```

¡Ahí está nuestro programa!

Cada variable va siendo llamada a medida que se necesita.

Al final, las variables son eliminada de la memoria con **{ N1 N2} PURGE.**

El programa anterior, tiene la desventaja de que produce error cuando los nombres N1 o N2 ya están siendo usados por otras variables.

2.1.2 Variables Locales.

En nuestros programas, Podemos usar un tipo de variables que son más rápidas que las variables globales y que, además, no tenemos que preocuparnos por eliminarlas ya que desaparecen de la memoria cuando el programa termina de ejecutarse. Estas variables son llamadas *Variables Locales*.

Otra muy importante ventaja de las variables locales es que pueden usar nombres que ya son usados por variables globales.

Las variables locales se crean con el comando → (la flecha verde, sobre la tecla del cero.) Para explicar su uso, rescribiremos el ejemplo anterior usando una estructura de variables locales.

```
<<
→ N2 N1
<< N2 N1 + N2 N1 –
N2 N1 * N2 N1 / >>
>>
```

Fíjese que lo que se ha hecho es colocar el comando → seguido por los nombres que queremos dar a las variables.

Obsérvese además que hemos escrito:

→ N2 N1

y no:

→ N1 N2

esto es porque las variables locales son creadas tomando los datos de la pila, desde arriba hacia abajo.

NOTA: Además de ser más rápido, este último programa ocupa menos memoria, ya que el anterior ocupa 86.5 bytes, mientras que este ocupa 72.5 bytes.

2.2 Notación Algebraica.

Si queremos un programa que tome dos números de la pila, y que divida su suma entre su diferencia, podemos escribir:

<< Y STO X STO X Y + X Y - / {X Y } PURGE >>

También se podría con:

<< → X Y << X Y + X Y - / >> >>

Otra forma sería:

<< DUP2 + SWAP ROT - / >>

Una cuarta forma de escribir el programa anterior es utilizando la llamada Notación Algebraica:

<< → Y X '(X-Y)/(X+Y)' >>

Muchos programadores de User Rpl prefieren utilizar sintaxis algebraica ya que los objetos algebraicos son fáciles de escribir y de leer.

La HP48 dispone de constantes simbólicas que pueden ser incluidas en las expresiones algebraicas:

π = 3.141592...
e= 2.718281...
i= √ -1

Escribamos un programa que calcule el área de un circulo(Area=πR^2) cuando el radio es dado en la pila:

El código sería:

<< → R 'π*R^2' →NUM >>

El comando →NUM fue incluido al final para forzar el cálculo numérico de la constante π. Esta situación también puede ser manejada cambiando el estado de los indicadores -2 y –3 del sistema. (ver pagina 4-7 de la *Guía de Usuario*)

2.3 Trabajando con subrutinas.

Escribamos un programa que produzca una melodía. Para esto, Hagamos primero un programa que genere un pitido de 1000 hz. durante 0.5 segundos:

<< 1000 .5 BEEP >>

y lo guardaremos con el nombre S1.

Luego hagamos otro, que produzca un pitido de 1200 hz. durante 0.5 segundos:

<< 1200 .5 BEEP >>

este último lo guardaremos con el nombre S2.

Finalmente, para producir nuestra melodía, solo tenemos que escribir un último programa (programa principal), desde el cual llamaremos a S1 y S2. Éste pudiera ser:

<< S1 S2 S1 S1 S2>>

a este último lo nombraremos PRIN.

Al ejecutar el programa principal (PRIN), se produce la melodía que queríamos. Se dice que S1 y S2 son *subrutinas* de PRIN.

Una subrutina puede ser sencilla o puede seguir dividiéndose en otras más pequeñas. Esto permite tener subrutinas relativamente sencillas, incluso si el programa principal es de gran tamaño

Otra ventaja es que podemos escribir varios programas que compartan una misma subrutina. Podemos por ejemplo escribir otro programa que utilice S1 y S2, pero que produzca una melodía distinta.

Un problema que se presenta al utilizar subrutinas es que las variables locales creadas en el programa principal no están al alcance de las subrutinas, es decir que si creas una variable local en el programa principal, no podrás utilizarla en las subrutinas.

Para solucionar este problema existe un tipo de variables llamadas *variables locales compiladas,* las cuales veremos a continuación.

2.4 Variables compiladas.

La HP48 ofrece una forma de utilizar en cualquier subrutina, las variables locales creadas en el programa principal. Para esto solo debes poner el símbolo ← como primer carácter de su nombre (si el nombre es X, escribimos ←X)

Ejemplo:

Programa Principal:

<< → ←X ←Y <<SUB1 SUB2 />>

>>

Primera subrutina(SUB1)

<< →X →Y + >>

Segunda subrutina (SUB2)

<< →X →Y - >>

2.5 El Debuguer.

La HP48 posee una herramienta que sirve para encontrar errores en nuestros programas. Si escribimos un programa y no está dando los resultados esperados, podemos utilizar el Debuguer para encontrar en qué parte del código se produce el error.

Esto es importante, ya que los errores son frecuentes durantes el desarrollo de programas y sin el Debuguer puede resultar una tarea difícil su localización.

Para depurar nuestro programa, lo primero es colocar su nombre en la pila ('NOMBRE')

Luego presionamos las teclas **PRG** y **NXT** y elegimos la opción de menú **RUN.**

Ahora solo debemos presionar DBUG para iniciar la depuración.

Si usted ha seguidos las instrucciones anteriores al pie de la letra, verá como se activa el indicador HALT, el cual indica que el programa ya esta cargado en memoria, listo para ser depurado.

El Debuger también puede iniciarse si al presionar DBUG, el programa se encuentra en el nivel **1:**

2:
1: <<programa>>

Para depurar el programa ya cargado solo debemos presionar SST sucesivamente para ir ejecutando el programa paso a paso. Una instrucción será ejecutada por cada vez que presiones SST..

La opción NEXT muestra la próxima instrucción, pero, sin ejecutarla

Ejercicio: Cargue el siguiente programa en el Debuger y ejecútelo paso a paso:

<< 2 1 + 5 * 7 / >>

La opción SST↓ sirve para introducirse en el código de una subrutina. Y la opción KILL cancela el proceso de depuración.

En el Ejercicio anterior, también se pudo iniciar el proceso de depuración escribiendo el comando HALT dentro del programa:

<< HALT 2 1 + 5 * 7 />>

Esto es útil para depurar subrutinas.

Supóngase que tenemos un programa con tres subrutinas y creemos que un error se esta produciendo después de la segunda subrutina, entonces, colocamos el HALT en la subrutina sospechosa y, al ejecutar el programa principal, éste se detendrá donde encuentre el HALT, habiendo ejecutado todas las instrucciones que le preceden.

Luego, solo tenemos que entrar al Debuger (PGR NXT RUN) y seguir el proceso presionando SST.

También podemos usar HALT en programas extensos sin subrutinas, colocándolo en el punto sospechoso. El programa se ejecutara hasta dicho punto.

Para desactivar el indicador HALT en cualquier momento, solo debemos ejecutar el comando KILL.

Curiosidades

El Directorio Oculto

Existe un directorio al cual no se puede tener acceso por los medios convencionales, ya que esta oculto en la memoria de la HP48. En este directorio la calculadora guarda ciertas informaciones importantes del sistema.

A continuación mostrare como entrar al directorio oculto, pero primero debo advertirle que no se debe borrar ni reorganizar las variables que allí se encuentren, ya que esto podría producir un "Try To Recover Memory"

Lo que se debe hacer para entrar al directorio oculto es evaluar un nombre nulo. El nombre nulo es un par de comillas simples vacías: ‘ ‘

Pero, ¿cómo podemos obtener el nombre nulo? Para esto se debe colocar en la pila el número hexadecimal `#15777h` y ejecutar el comando SYSEVAL. El siguiente programa nos permitirá entrara al directorio oculto:

<< HOME `#15777h` SYSEVAL EVAL>>

Se puede ocultar todo el contenido de un directorio (excepto el directorio HOME) Para esto solo se debe entrar al directorio a ocultar y ejecutar el siguiente programa:

<<{ } `#15777h` SYSEVAL EVAL STO>>

Para ejecutar las variables escondidas solo tenemos que escribir su nombre.

Para que las variables vuelvan a aparecer ejecute el siguiente programa:

<< `#15777h` SYSEVAL PURGE>>

CAPÍTULO III. Cadenas y Listas

3.1 Trabajando con Textos

En esta sección, aprenderemos a crear programas que trabajen con cadenas de caracteres. Las cadenas se conocen porque siempre están entre comillas dobles. Los siguientes son ejemplos de cadenas:

1) "HOLA"
2) "1 2 3"
3) "Esto es una Cadena"

El lector ya conoce el comando MSGBOX, el cual toma una cadena de la pila y la presenta en un recuadro. A continuación veremos otros comandos que sirven para manipular cadenas:

SIZE : Toma una cadena del nivel 1: y devuelve la cantidad de caracteres y espacios que la forman.

+ : Toma dos cadenas de la pila y las une. Si tenemos por ejemplo :
2: "Version "
1: "HP48R"
al ejecutar << + >> resultara:

1: "VersionHP48R"

Nótese que el comando + no agrega espacio entre las dos cadenas, para esto podemos sustituir "Versión" por "Versión ".

El comando + funciona también cuando tenemos una cadena y algunos otros tipos de objeto. Aprovechando esto, escribamos un programa que tome un número de la pila y nos diga cual es el resultado de sumarle 17:

<< 17 + " El Resultado es " SWAP + MSGBOX >>

OBJ→: Divide una cadena en las palabras que la componen. Si tenemos en la pila el texto " HOY ES SABADO" y ejecutamos OBJ→, el resultado será:

3: 'HOY'
2: 'ES'
1: 'SABADO'

Hagamos un programa que nos diga la cantidad de palabras de una cadena:

<<

→ CADENA << CLEAR CADENA OBJ→ DEPTH "PALABRAS" + MSGBOX CLEAR >>

>>

Debe tenerse mucho cuidado al usar el comando CLEAR, ya que éste elimina todos los elementos que haya en la pila. Por eso, en el programa anterior, antes del CLEAR hemos guardado la cadena como variable local.

→STR : Toma un objeto del nivel 1: y lo convierte en una cadena.

HEAD : Extrae el primer objeto de una cadena.

TAIL : Elimina el primer caracter de una cadena.

POS : devuelve la posición en que aparece por primera vez un carácter.

Si tenemos:

2: "REPUBLICA DOMINICANA"
1: "A"

al ejecutar POS obtendremos:

1: 9

esto es porque el carácter "A" aparece por primera vez en la posición 9 de izquierda a derecha.

Ejercicio: Escriba un programa que tome una letra de la pila y diga su posición en el alfabeto.

NUM y **CHR**

A cada caracter de la HP48 le corresponde un número único. El número del carácter "A", por ejemplo es el 65, el de "B" es 66 y el número de "7" es 55.

Si tenemos un carácter en la pila (entre comillas dobles) y ejecutamos el comando NUM, obtendremos el número de dicho carácter.

Si, por el contrario, tenemos el número y queremos el carácter correspondiente, ejecutamos el comando CHR.

(a) Investigue los caracteres correspondientes a los siguientes números:

48___	88___	120___	135___
49___	89___	121___	140___
57___	90___	122___	223___

(b) Escriba un programa que transforme en una letra mayúscula en minúscula.

SUB : Dado una cadena en 3: y dos número X y Y en los niveles 1: y 2:, devuelve la parte de la cadena que esta entre las posiciones X y Y

Ejercicio: Escriba un programa que tome una cadena de la pila y devuelva sus últimos tres caracteres.

REPL : Toma una cadena de 3: y remplaza el carácter cuya posición es dada en 2: por un carácter dado en 1:

Para aclarar esto, veamos un ejemplo. Escribamos un programa que remplace el tercer carácter de una cadena cualquiera por la letra M. El código será:

<< 3 "M" REPL >>

Si la cadena dada tiene menos de tres caracteres, el comando RPL simplemente colocara la M al final.

Como hemos podido apreciar, existen muchos comandos para manipular cadenas de caracteres.

3.2 Trabajando con Listas.

Una lista es una colección de objetos, colocados entre los delimitadores { y }.

Una lista puede contener cualquier cantidad de objetos, y de cualquier tipo. Los siguientes son ejemplos de lista:

{ 1 2 -7 9 }

{ "TEXTO" 3.1416 (2,3) }

{ << SWAP 1 + * >> 'PROG1' }

También podemos tener una o más listas dentro de otra:

{ {1 2 3} "texto" }

{ { a b c d } { { X1 Y1} { X2 Y2 } } }

Las listas pueden llegar a ser muy útiles para los programadores ya permiten tener varios objetos en uno solo.

3.2.1 Trabajar con listas es parecido a trabajar con cadenas.

Existen algunos comandos de cadenas que funcionan también con las listas:

SIZE : devuelve el número de elementos de una lista

OBJ→ : Descompone una lista en sus elementos. También devuelve al nivel 1: el número de elementos de la lista .

HEAD : Devuelve el primer elemento de una lista.

TAIL : Elimina el primer elemento de una lista.

SUB : Extrae una parte de una lista. Este comando requiere:

3: La lista
2: posición inicial
1: posición final

Ej: El siguiente programa extrae los 3 primero elementos de una lista:

<< 1 3 SUB >>

REPL : Remplaza un elemento de lista por otro. Este comando requiere:

3: La Lista
2: Posición del objeto a remplazar.
1: Objeto sustituto.

Ej: El siguiente programa remplaza el tercer elemento de una lista por 3.1416:

<< 3 3.1416 REPL >>

+ : Concatena listas.

Ej:

2: { a b }
1: { 1 7 9 }

POS: Devuelve la posición en que aparece por primera vez un objeto.

Ej:

2: { 1 3 5 3 7 }
1: 3

3.2.2 otros comandos para listas.

SORT : Clasifica los elementos de una lista en orden ascendente.

Ej:

1: { 1 3 2 7 5}

REVLIST : Invierte los elementos de una lista

Ej:

1: { a b c d e }

Escribamos un programa que invierta el orden de las variables del menú VAR:

<< VARS REVLIST ORDER >>

ΣLIST : Suma todos los elementos de una lista.
Ej1:
1: { 3 7 2 }

Ej2:
1: { 5 T }

ΠLIST : Multiplica todos los elementos de una lista.

Ej1:
1: { 2 3 5 }

Ej2:
1: { 5 T }

ADD : Suma los elementos correspondientes de dos listas.

Ej:

2: { 5 8 0 }
1: { 1 3 27 }

∆LIST : Lo que hace este comando es lo siguiente:

Si en el nivel 1: tenemos una lista como:

{ a b c d }

obtendremos otra lista:

{ b-a c-b d-c }

Ej1:

1: { 2 5 12 }

Ej2:

1:{ 2 5 12 17 }

LIST→ : Sirve para descomponer una lista en sus elementos. Además nos dice el número de la lista (nivel 1:)

Ej:

4:
3:
2:
1: { a b c }

→LIST : Hace lo inverso al comando anterior: Toma una serie de objetos y su cantidad en 1: y crea una lista de dichos objetos.

Ej:

4:	a	→	**4:**
3:	b		**3:**
2:	c		**2:**
1:	3		**1:** { a b c }

GET : Dado una lista (en nivel 2:) y la posición de uno de sus elementos (en nivel 1:) extrae dicho elemento de la lista.

El siguiente programa toma una lista de la pila y extrae su tercer elemento.

<< 3 GET >>

Investigue los siguientes comandos:

1) GETI
2) PUT
3) PUTI
4) DOLIST
5) DOSUB
6) STREAM

Curiosidades

Emu48

Existe un programa gratuito para la emulación de la HP48 y otros modelos de calculadoras HP en Sistemas Operativos Windows. Este programa se llama Emu48 y es obra de los señores Sebastián Carlier y Christoph GieBelink

El empleo del emulador brinda muchísimas ventajas durante el desarrollo de programas. Entre sus ventajas podemos mencionar la mayor velocidad y mayor tamaño de pantalla de que se dispone. También se puede emular una tarjeta RAM insertable y disponer de buen espacio en memoria para trabajar. Otra de las ventajas del emulador es que podemos guardar el estado de la calculadora y volverlo a cargar en cualquier momento, lo que es especialmente importante cuando se trabaja en System Rpl o en Assembler.

CAPÍTULO IV. Estructuras Condicionales

3.1 Estructuras Condicionales

En este capítulo aprenderemos a hacer que nuestros programas tomen algunos tipos de decisiones.

Escribamos un programa que tome un número de la pila y si es mayor de 18 haga un beep. Para esto usaremos la estructura IF...THEN...END (si... entonces...fin). El programa será:

<< 18 IF > THEN 1000 .5 BEEP END >>

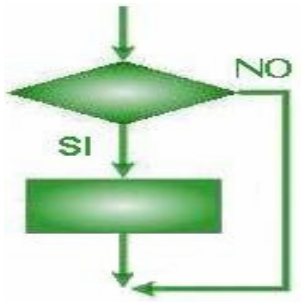


Diagrama de Flujo para una Estructura IF...THEN...END

ANALICEMOS:

Lo primero que hace el programa es enviar un 18 a la pila. Entonces se tiene:

1: Número dado
2: 18

Luego usa > para verificar si el número dado es mayor que el 18. En caso afirmativo, se ejecutan las instrucciones que están entre THEN y END.

Otras pruebas que podemos usar en nuestros programas son:

==	Igual que
≠	Diferente de
<	Menor que
≤	Menor o igual que
≥	Mayor o igual que

(Presionar PRG y luego entra al menú TEST)

Estas pruebas toman dos números de la pila. Si el resultado de la prueba es verdadero, se envía un 1 a la pila; si es falso, se envía un 0.

Si el comando THEN encuentra un 1, se ejecutan las instrucciones entre THEN y END. Pero, si encuentra un 0, las salta.

Nota: Cualquier número real diferente de 0 es considerado como verdadero.

Obsérvese que, al realizar la prueba, los números comparados son eliminados de la pila. En algunos casos, los números dados se utilizaran posteriormente y por esto deben ser guardados como variables o simplemente duplicados.

Rescribamos el ejemplo anterior para que también se produzca un pitido cuando la prueba sea falsa, pero éste será de 900 hertz. Para esto utilizaremos la estructura IF...THEN...ELSE...END.

El código será:

```
<<
18 IF > THEN  1000 .5 BEEP
          ELSE    900 .5 BEEP
     END
>>
```

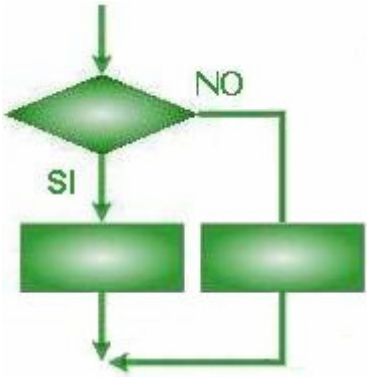


Diagrama de Flujo para una Estructura IF...THEN...ELSE...END

El ejemplo anterior puede ser mejorado si se escribe:

```
<<
 18 IF > THEN  1000
          ELSE    900
     END  .5 BEEP
>>
```

Lo que acabamos de hacer se llama optimizar el código.

Cuando escribamos nuestros programas debemos tratar de que esté en su condición óptima; debemos revisarlos para ver si podemos **reducir su tamaño** o **hacer que funcione más rápido**.

3.2 Formas compactas.

¿Qué le parecería tener los comandos IF, THEN, ELSE y END compactados en uno solo? O sea, poder crear una estructura condicional utilizando un solo comando en lugar de los cuatro anteriores. Esto es exactamente lo que hace el comando IFTE.

Veamos cómo funciona:

Si tenemos el programa:

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

podemos rescribirlo de forma abreviada utilizando IFTE:

```
<<
0 <  "NEGATIVO"  "POSITIVO" IFTE
>>
```

Como se puede apreciar, lo primero en el código es la prueba (0 <). Luego, la opción para el caso de que la prueba sea verdadera. Seguido se coloca la opción para el caso de que la prueba sea falsa y finalmente el IFTE.

Veamos otro ejemplo un poco más interesante:

Si tenemos el programa:

```
<<
DUP2  IF > THEN 3  *  -
                ELSE 2 / +
      END
>>
```

podemos rescribirlo como:

```
<<
DUP2  >  << 3  *  - >> << 2  /  +>>
IFTE
>>
```

Pero este último ocupa más memoria que su forma no compacta (7.5 bytes), lo cual nos dice que la notación compacta no siempre es la mejor opción.

> Existe también una forma compacta para IF...THEN...END. Dicha forma compacta es IFT

3.2 Estructura CASE

Existe un tipo de estructura condicional más poderosa que las estudiadas hasta este momento. Esta es la estructura CASE. Dicha estructura permite crear programas con más de dos opciones condicionales.

Veamos un programa que toma un número de la pila (entre cero y nueve) y desplegué un texto con su nombre:

```
<< → N << CASE
        N 0 = THEN "CERO" END
        N 1 = THEN "UNO" END
        N 2 = THEN "DOS" END
        N 3 = THEN "TRES" END
        N 4 = THEN "CUATRO" END
        N 5 = THEN "CINCO" END
        N 6 = THEN "SEIS" END
        N 7 = THEN "SIETE" END
        N 8 = THEN "OCHO" END
        N 9 = THEN "NUEVE" END
        "DATO INCORRECTO"
  END  MSGBOX >>
```

El ejemplo anterior talvez podría parecer un poco complicado, pero no es así. Después del análisis siguiente todo quedara claro

ANÁLISIS.

Lo primero que hace es guardar el número dado como variable local de nombre N. Luego se prueba si N es igual a cero y en caso afirmativo se envía el texto "CERO" a la pila y se salta al MSGBOX.

Si N no es igual a cero entonces se prueba si es igual a uno. En caso de que sea igual a uno se envía el texto "UNO" a la pila y se salta al MSGBOX.

En caso de que N tampoco sea igual a uno, se prueba si es igual a dos, y así sucesivamente.

Entre los dos END del final del código se encuentra la llamada *clausura por defecto*. Esta es la que se ejecutara en caso de que todas las pruebas anteriores hayan sido falsas. Esta clausura es opcional por lo que si se quiere puede dejarse ese espacio vació.

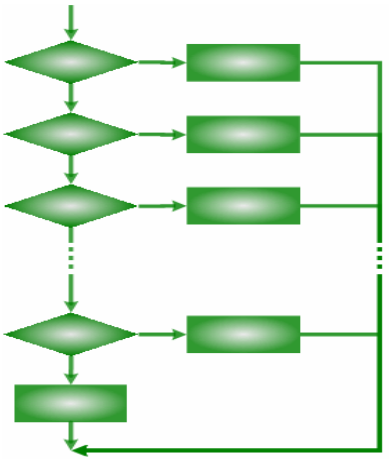


Diagrama de Flujo para una Estructura CASE...END

La estructura CASE es una de las más potentes del lenguaje User Rpl. El programador puede crear cualquier cantidad de opciones, estando únicamente limitado por la cantidad de memoria disponible.

3.3 Estructuras de detección de Errores.

Qué pasa si ejecutamos el siguiente programa sin que haya ninguna dato en la pila:

<< SWAP - >>

Si hacemos eso, se producirá un error que generara un pitido y desplegara un mensaje en la pantalla

En algunos casos, se quiere que, al ocurrir un error, nuestro programa se comporte de determinada forma. Podemos, por ejemplo, querer que nuestro programa le indique al usuario el orden exacto en que debe introducir los datos.

Existe un tipo especial de estructuras que permiten dar instrucciones a nuestros programas para ser ejecutadas en el preciso momento en que ocurre un error. Estas son las llamadas **estructuras de detección de Errores**. Dichas estructuras son:

IFERR...THEN...END

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

Obsérvese el parecido que tienen con las estructuras condicionales que estudiamos al principio. En realidad trabajan de manera análoga.

Se debe escribir nuestro programa entre IFERR y THEN. Las instrucciones a ejecutarse en caso de que ocurra un error se colocaran entre THEN y END en la primera estructura o entre THEN y ELSE en la segunda.

La segunda estructura permite una opción entre ELSE y END, la cual se ejecutara en caso de que no se produzca ningún error. Esto puede llegar a ser muy útil en algunos casos.

El ejemplo del principio de esta sección puede rescribirse como:

```
<< IFERR SWAP - THEN
  "FALTAN DATOS" MSGBOX
  END >>
```

3.4 Comandos relacionados con errores.

En el apéndice B-1 de la *Guía de Usuario*, se encuentra una lista de los distintos errores que pueden ocurrir en la HP48. También se indica las causas de que se hayan producido y un número único correspondiente a cada tipo de error. El número #C14h, por ejemplo, corresponde al mensaje Low Battery.

El comando DOERR hace que se despliegue un mensaje de error dado su número. Escribamos un programa que despliegue el mensaje Low Battery:

```
<< #C14h DOERR >>
```

El último error ocurrió es guardado en memoria automáticamente. El comando ERRM devuelve el mensaje del último error y el comando ERRN devuelve su número.

El comando ERR0 borra de la memoria todos los datos relacionados con el último error.

Curiosidades

El Saturno

La arquitectura de procesador usada en la HP48 recibe el nombre de Saturno. Este CPU de 4 bits fue especialmente diseñado por Hewlett Packard para bajo consumo de energía y excelente desempeño en matemáticas. El procesador Saturno apareció por primera vez en la HP 71B y desde entonces se ha usado de diferentes maneras en otros modelos de calculadoras HP. Su implementación en la HP48 opera a unos 4MHz y es fabricado por NEC en Japón.

CAPÍTULO V. Entrada y Salida de Datos

Los programas reciben información suministrada por el usuario y luego la procesan y producen ciertos resultados.

La HP48 ofrece gran cantidad de formas para solicitar datos a los usuarios. También ofrecen muchas otras formas de presentar los resultados de procesar esos datos.

4.1 Entrada de datos.

Hasta ahora nuestros programas toman los datos directamente de la pila. Esto puede ocasionar problemas cuando el usuario no recuerda el orden en que debe introducirlos. Por suerte existen métodos más eficientes. Comenzaremos estudiando los más simples:

4.1.1 Método INPUT

Escribamos un programa que calcule el área de un rectángulo y utilice un entorno INPUT para pedir los datos. El código es:

```
<<
"INTRODUZCA DATOS"
{":LARGO":
  ":ANCHO:"  8}
INPUT OBJ-> *
>>
```

Córralo en su HP48 para que vea cómo funciona. Utilice el DEBUGER.

Como verá, lo primero es el texto del encabezado y luego las etiquetas para cada dato solicitado.

El número 8 indica la posición del cursor. Si lo sustituyes por un número mayor, el cursor aparcera más a la derecha y si lo remplazas por uno menor, aparecerá más a la izquierda.

Si queremos que se active el teclado alfabético, solo hay que escribir α después del 8. Esto es importante cuando se requiera la introducción de textos.

El **OBJ→** separa los datos introducidos para que pueda entrar en acción el *****

> Si queremos que el cursor sea el de reemplazo, solo debemos sustituir el 8 por -8

4.1.2 Método TMENU

El comando TMENU, permite crear un menú temporal que podemos utilizar para la introducción de datos.

Veamos cómo sería el código para un menú que pida tres datos A, B y C y que luego se calcule :

(A+B)/(A-C)

```
<<
{{ "→A" << A STO>> }
 { "→B" << B STO >> }
 { "→B" << C STO >> }
 { "OK" << CONT >> }
  {   }
{"EXIT" <<{A B C} PURGE >>}}
TMENU HALT A B + A C - /
{A B C} PURGE
>>
```

Cópielo en su HP y observe cómo funciona. El análisis se deja como ejercicio al lector.

4.1.3 El método PROMPT

Éste es uno de los métodos más simples. El PROMPT toma una cadena y la muestra como instrucciones en el área de estado. Luego detiene la ejecución del programa para que el usuario introduzca los datos de acuerdo a las instrucciones. Tras hecho esto, El usuario debe presionar CONT ( , ON) para continuar la ejecución del programa.

Veamos un ejemplo de un programa que toma tres números X, Y, Z, de la pila y evalúa la ecuación (2X+Y)/Z.

```
<< "3: Z  2: Y  1: X"  PROMPT
  2 * + SWAP / >>
```

Obsérvese que el texto **"3: Z 2: Y 1: X"** le indica al usuario el orden exacto en que debe introducir los datos.

Después de colocar los datos en la pila, el usuario presiona CONT para que los cálculos se realicen.

Por ahora solo utilizaremos los tres métodos anteriores para entrada de datos. En otros capítulos veremos otros métodos como son el INFORM y el CHOOSE.

4.2 Salida de datos.

4.2.1 ...DISP...FREEZE

Este método es muy usado Ya que ocupa poca memoria y permite mostrar datos en lugares distintos de la pantalla.

El comando DISP toma un texto y lo presenta en una posición indicada. Veamos un ejemplo:

```
<<
"CREADO POR MARK "
2 DISP
>>
```

El número 2 antes del DISP representa la posición. Este número puede variar del 1 al 7 donde el 1 es la parte superior de la pantalla; y el 7, la parte inferior.

Para que el mensaje no se desaparezca utilizaremos el comando FREEZE

```
<<
"CREADO POR MARK "
2 DISP  3 FREEZE
>>
```

1 FREEZE: Congela el área de estado.
2 FREEZE: Congela la pila y la línea de comando.
3 FREEZE: Congela la pantalla completa.

Si queremos eliminar el fondo, utilizamos el comando CLLCD

```
<<
CLLCD
"CREADO POR MARK "
2 DISP  3 FREEZE
>>
```

También podemos mostrar varios mensajes al mismo tiempo. Solo se requiere una lista con los mensajes y otra con las posiciones:

```
<<
CLLCD
{ "X = 5" "Y = 7" " Z = 12"}
{1 2 3}  DISP
3 FREEZE
>>
```

Caracteres de Fondo.

Escribamos un programa que produzca la siguiente presentación.


```
<< "¤¤¤¤¤¤¤¤¤¤¤¤
    ¤¤¤¤¤¤¤¤¤¤¤¤"
    { 1 2 3 4 5 6 7 }  DISP
    "Fundamentos de User Rpl"
    MSGBOX  >>
```

4.2.2 Mostrando dibujos

Como sabemos, la HP48 posee una herramienta para la creación y edición de dibujos, a la cual accedemos al presionar la tecla  .

Vamos a utilizar el comando ->LCD para escribir un programa que tome un

Gráfico de la pila y lo muestre:

```
<< →LCD 3 FREEZE >>
```

CAPÍTULO VI. Bucles

6.1 La Recursividad.

Vamos a crear un programa que haga conteo regresivo. O sea que si pones, por ejemplo, un 500 en la pila, el programa contara 499, 498, 497,...,1. El código será:

```
<< DUP 0 IF >
THEN 1 - DUP 3 DISP
 RECX END >>
```

Para que funcione, el programa debe ser guardado con el nombre RECX. Luego coloca un número en la pila (por ej. 500) y ejecútalo.

ANÁLISIS

EL programa verifica si el número dado es mayor que cero. En caso afirmativo le resta 1 y muestra el resultado en la pantalla. Luego el programa **se llama a si mismo** por su nombre (RECX), por lo que la prueba vuelve a repetirse. El proceso continúa hasta que el número en la pila se vuelve cero.

La Recursividad es un método que se puede utilizar para resolver problemas que implican la realización de muchos pasos idénticos en sucesión.

Al utilizar Recursividad se debe tener cuidado de no caer en un proceso que se repita indefinidamente.

Ejercicio: Escriba un programa que calcule el factorial de un número dado utilizando recursividad.

6.2 Estructuras de Bucle.

Las estructuras iterativas o de bucle son un tipo muy especial de estructuras que permiten que un programa ejecute una serie de comandos varias veces. Existen dos tipos de estructuras de bucle: Las definidas y las Indefinidas.

En esta sección conoceremos los bucles definidos, los cuales se usan cuando queremos que las instrucciones se repitan una cantidad especificada de veces.

6.2.1 Estructura START...NEXT

Toma dos números (inicio y final) de la pila y ejecuta todas las instrucciones que estén entre START y NEXT, una cantidad de veces igual a la diferencia entre los dos números dados.

Para aclarar esto, veamos un ejemplo: Escribamos un programa que multiplique cuatro números dados en la pila. Para esto, podemos escribir:

```
<< * * * >>
```

y efectivamente obtendríamos el resultado esperado. Pero, ¿qué pasaría si necesitáramos un programa que multiplique 50 números?

Si utilizamos el método anterior en este último caso, obtendríamos un programa bastante extenso.

Una solución inteligente para este tipo de problemas sería utilizar una estructura de bucle:

```
<< 1 50 START * NEXT >>
```

Lo que hace esta estructura es lo siguiente: Crea una variable con el primer número (1), llama *contador del bucle,* y la incrementa en uno cada vez que realiza un ***.** El proceso se detiene cuando la variable llega a ser igual al segundo número (50).

Veamos otro ejemplo. Escribamos un programa que genere una serie de pitidos cuya frecuencia se vaya incrementándose desde 100 hasta 900 hertz.

```
<<
0
1  9  START
DUP  100  +  0.5  BEEP
NEXT  DROP
>>
```

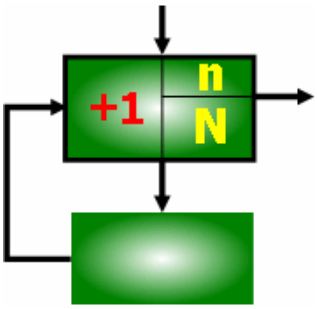


Diagrama de Flujo para una estructura START...NEXT...END

6.2.2 Estructura FOR...NEXT

Este bucle es muy parecido al START NEXT, la diferencia está en que con este último podemos usar el contador**.** Veamos un ejemplo. El siguiente programa hace un conteo desde 1 hasta 100:

```
<<
1 100 FOR  C
 C 3 DISP NEXT
>>
```

La C a la derecha de FOR es el nombre que yo he dado al contador.

Se llama al contador por su nombre (C) y se hace un 3 DISP.

Se Puede usar cualquier nombre para el contador

```
<<
1   10   FOR FREC
100 FREC * .5 BEEP NEXT
>>
```

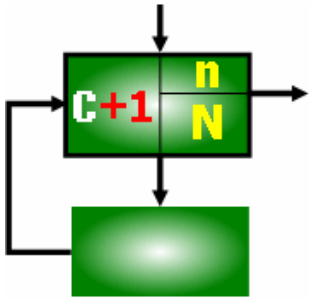


Diagrama de Flujo para una estructura FOR...NEXT...END

6.2.3 Estructura FOR...STEP

Obsérvese que, en el ejemplo anterior, la multiplicación por 100 es necesaria porque el contador se va incrementando de uno en uno:

1, 2, 3,...,10

La multiplicación por 100 podría evitarse si el incremento fuera de cien en cien:

100, 200, 300,..., 1000

Para esto solo hay que sustituir NEXT por STEP y colocar el incremento que queramos para el contador justo antes del STEP. Esto es:

```
<<
100 1000 FOR FREC
FREC .5 BEEP
100 STEP
>>
```

El cambio en los valores inicial y final no debe sorprendernos ya que ahora el contador ira desde cien hasta mil.

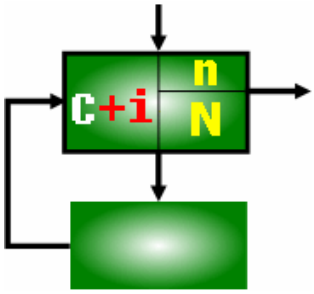


Diagrama de Flujo para una estructura FOR...STEP...END

De manera similar podemos sustituir START...NEXT por START STEP:

```
<<
inicio fin START...incremento STEP
>>
```

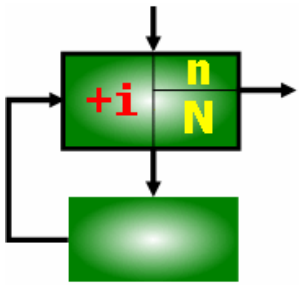


Diagrama de Flujo para una estructura START...STEP...END

Decrementos

¿Cómo sería el programa anterior si quisiéramos que la frecuencia se fuera reduciendo desde 1000 hasta 100?

```
Una solución sería:
<<
100  1000  FOR FREC
1100  FREC  - .5  BEEP
100  STEP
>>
```

Pero el comando STEP acepta incrementos negativo por lo que es más fácil escribir:

```
<<
1000  100  FOR FREC
FREC  .5 BEEP
-100  STEP
>>
```

Ejercicios

(a) Escriba un programa que calcule:

1) 1 + 1/2 + 1/3 + ... + 1/99
2) 2^1 + 2^2 + 2^3 + ... + 2^50
3) 1/99 + 2/98 + 3/97 + ... + 99/1

(b) Escriba un programa que tome una cadena de caracteres del nivel 1: y cambien sus letras mayúsculas por minúsculas.

(c) Utilizando el comando DISP y una estructura de bucle, escriba un programa que muestre un texto que vaya ascendiendo en la pantalla.

(d) Escriba un programa que tome cinco números de la pila, y si todos son pares, los sume.

(e) Combinando los comandos CHR y DISP, escriba un programa que muestre todos los caracteres de la HP

(f) Escriba un programa que tome un texto de la pila y extraiga todas las vocales que contenga.

(g) Escriba un programa que genere un pitido cuya frecuencia varíe sinusoidalmente.

TEMA VII. Vectores y Matrices

6.1 Trabajando con Sistemas.

Cuando hablamos de sistemas, nos referimos tanto a vectores como a matrices. La HP48 ofrece un extenso repertorio de comandos para trabajar con sistemas. En este capítulo conoceremos algunos.

OBJ→ : descompone una matriz en sus elementos. También devuelve las dimensiones de la matriz en una lista {filas Columnas}

→DIAG : Toma una matriz y devuelve su vector de diagonal.

→ROW : Descompone una matriz en sus vectores de fila.

→COL : Descompone una matriz en sus vectores de columna.

ROW→ : Dado varios vectores en la pila y su cantidad en el nivel 1:, crea una matriz con los vectores dados. Los vectores son tomados como filas de la matriz.

```
4:  [1 2 3]                1:  [[1 2 3]
3:  [4 5 7]        →           [4 5 7]
2:  [0 1 5]                    [0 1 5]]
1:        3
```

COL→ : Es similar al comando anterior, solo que los vectores son tomados como columnas.

DET : devuelve el determinante de una matriz cuadrada.

TRN : transpone una matriz dada

ABS : dado una matriz o un vector, devuelve la raíz cuadrada de la suma de los cuadrados de sus elementos.

ROW− : dado una matriz en 2: y un número en 1:, extrae la fila que se encuentra en la posición correspondiente a ese número.

```
2:  [[2 5]                 2:  [[3 1]
    [3 1]          →           [0 4]]
    [0 4]]             1:  [2 5]
1:      1
```

COL− : Es similar al comando anterior, solo que extrae una columna.

ROW+ : Inserta una o más filas nuevas en una matriz.

```
3: [ [1 2]                 1: [ [1 2]
     [3 4] ]                   [0 7]
2:  [0 7]          →           [3 4] ]
1:      2
```

COL+ : Similar al anterior, pero con columnas.

GET : extrae un elemento determinado de una matriz.

```
2: [[1 6 0][7 3 9]]   →   2:
1:  4                     1:   7
```

En el ejemplo anterior se extrajo el cuarto elemento de la matriz en 2:

PUT : sustituye un elemento de una matriz por otro. En el nivel 2 debe haber una lista {fila columna}

```
3: [[1 6 0][0 3 9]]
2:  {2  1}          →   2:
1:   7                  1: [[1 6 0][7 3 9]]
```

Ejercicio: Investigue los siguientes comandos.

1)	CON	6)	CSWP
2)	IDN	7)	RDM
3)	RANM	8)	RCI
4)	→DIAG	9)	RCIJ
5)	RSWP	10)	TRACE

6.2 Matrices complejas.

La HP48 también ofrece comandos para manipular matrices complejas.

C→R : Toma una matriz compleja y la descompone en dos matrices reales.

RE : extrae la parte real de una matriz compleja.

IM : extrae la parte imaginaria de una matriz compleja.

R→C : Dado dos matrices reales en la pila, las combina en una matriz compleja.

CONJ : conjuga cada uno de los elementos de una matriz compleja

En relación a vectores, también son útiles los comandos siguientes:

DOT : Dado dos vectores A y B de igual dimensión, calcula su producto escalar A.B.

CROSS : Dado dos vectores A y B, calcula su producto vectorial AxB.

6.2 Trabajando con polinomio.

Como sabemos, un polinomio simbólico como:

$$X^3+4\cdot X^2-7\cdot X+9$$

también puede escribirse como un vector de sus coeficientes, o sea:

[1 4 –7 9]

Para encontrar todas las raíces de un polinomio existe el comando **PROOT**, el cual solo requiere el vector de coeficientes en 1:

Si queremos, por ejemplo, encontrar las raíces de:

$$X^2-3\cdot X+2$$

simplemente ponemos en la pila el vector [1 –3 2] y ejecutamos el comando PROOT.

El resultado será el vector [1 2] donde 1 y 2 son las raíces buscadas.

Y que tal si quisiéramos saber cuanto vale

$$X^2-3\cdot X+2$$

cuando x = -7

Para esto, colocamos el vector de coeficientes en 2: y el –7 en 1:. Luego ejecutamos el comando **PEVAL**.

2: [1 –3 2]
1: 7 → **2:**
1: 72

Pero también puede ocurrir que conozcamos las raíces de un polinomio y queramos saber cuales son los coeficientes. En este caso colocamos el vector de raíces en 1: y ejecutamos el comando **PCOEF**.

CAPÍTULO VIII. Cero y Uno

7.1 Operaciones Lógicas.

Como se pudo ver en capítulos anteriores, la HP48 puede hacer comprobaciones o tests utilizando las funciones siguientes:

==	Igual que
≠	Diferente de
<	Menor que
>	Mayor que
≤	Menor o igual que
≥	Mayor o igual que

Vimos, por ejemplo, que la función > prueba si un número en el nivel 2: es mayor que otro en 1:. En caso de que la prueba sea verdadera, se enviará un 1 a la pila; pero si es falsa se enviará un 0.

Se dice que verdadero y falso son valores lógicos.

La HP48 ofrece una serie de operaciones para que los programadores puedan trabajar con valores lógicos:

NOT : Es una operación que sirve para negar el valor lógico de una prueba. O sea, si el resultado es verdadero lo convierte en falso y si es falso, lo convierte en verdadero**.**

Si evaluamos por ejemplo

<< 3 5 < >>

el resultado será verdadero (1).

Pero si evaluamos:

<< 3 5 < NOT >>

el resultado será falso (0)

A	A NOT
0	1
1	0

NOT es la única operación que utiliza un solo valor lógico.

AND : Esta operación toma dos valores lógicos de la pila y si alguno de ellos es falso, da como resultado un cero. Dicho de otra forma, la operación AND solo será verdadera cuando los dos valores dados sean verdaderos**.**

A	B	A B AND
0	0	0
0	1	0
1	0	0
1	1	1

La operación AND es también conocida como producto lógico
A*B

OR : Toma dos valores lógicos de la pila y si alguno de ellos es verdadero, da como resultado un 1. Dicho de otra forma, la operación OR solo será falsa cundo los dos valores dados sean falsos.

A	B	A B OR
0	0	1
0	1	1
1	0	1
1	1	0

La operación OR es también conocida como suma lógica
A+B

XOR : Toma dos valores lógicos de la pila y si son iguales, da como resultado un 0. Dicho de otra forma, la operación XOR solo será verdadera cundo los dos valores dados sean diferentes.

A	B	A B XOR
0	0	0
0	1	1
1	0	1
1	1	0

La operación XOR es en realidad una combinación de otras operaciones
A ⊕ B

Es importante mencionar que las operaciones lógicas toman cualquier número diferente de cero como valor lógico verdadero

Solo el cero es considerado como falso.

7.2 Los Indicadores o Flags.

Como sabemos, la HP48 posee una especie de interruptores que sirven para controlar el modo en que ha de funcionar. Estos interruptores son los llamados *indicadores del sistema* (ver pagina 4-7 de la *Guía de Usuario*)

El comando **SF** sirve para activar cualquier indicador y el comando **CF** sirve para desactivarlo. Estos comandos solo requieren el número del indicador en la pila. (ver apéndice D de la *Guía de Usuario*)

Escribamos un programa que active la visualización del reloj:

```
<< -40 SF >>
```

También podemos activar o desactivar varios indicadores a la vez. Para Activar los indicadores –1, -2 y –3, el código sería:

```
<< {-1 -2 -3} SF >>
```

Otros comandos útiles son:

FS? : devuelve 1 si el indicador está activado o un 0 si está desactivado.

FC? : devuelve un 1 si el indicador está desactivado o un 0 si está activado.

FS?C : devuelve un 1 si el indicador está activado o un 0 si no lo está; luego desactiva el indicador.

FC?C : devuelve un 1 si el indicador está desactivado o un 0 si está activado; luego desactiva el indicador.

La HP48 cuenta con 128 indicadores de los cuales solo 64 son del sistema. Los otros 64 pueden ser usados por los programadores para controlar el funcionamiento de sus programas. Estos últimos son conocidos como *indicadores de usuario*.

Los indicadores pueden llegar a ser muy útiles ya que su estado no es afectado por el apagado o encendido de la calculadora.

Los números del 1 al 64 corresponden a indicadores de usuario. Se usan números positivos para diferenciarlos de los del sistema.

7.3 Los tipos de Objetos de la HP48.

Como sabemos, la HP48 puede trabajar con diversos tipos de objetos, tales como: números reales y complejos, cadenas de caracteres, objetos gráficos, listas, etc.

Es importante saber que existe un número asociado a cada tipo de objeto. Las cadenas de caracteres, por ejemplo, son objetos tipo dos y los números reales son objetos tipo cero.

En la siguiente tabla se muestran los diferentes tipos de objetos y su número asociado:

OBJETO	TIPO
Número Real	0
Número complejo	1
Cadena	2
Matriz Real	3
Matriz compleja	4
Lista	5
Nombre Global	6
Nombre local	7
Programa	8
Algebraico	9
Entero Binario	10
Gráfico	11
Objeto Etiquetado	12
Objeto Unidad	13
Nombre XLIB	14
Directorio	15
Librería	16
Backup	17
Función Incorporada	18
Comando Incorporado	19
Binario (Sistem Rpl)	20
Real Extendido	21
Complejo Extendido	22
Caracter	24
Code	25
Librería Data	26
Objeto External 1	27
Objeto External 2	28
Objeto External 3	29
Objeto External 4	30

Algunos de los objetos de la tabla anterior corresponden al mundo del System Rpl y otros al del ML.

Si tenemos un objeto cualquiera en a pila y queremos saber el número del tipo a que pertenece, solo hay que ejecutar el comando **TYPE.**

Si el objeto está guardado como variable, solo hay que colocar el nombre en la pila y ejecutar **VTYPE**.

El comando TYPE puede ser usado para crear programas que actúen de acuerdo al tipo de objeto suministrado

CAPÍTULO IX. Bucles (Última parte)

En capítulos anteriores, estudiamos los bucles definidos, los cuales se ejecutan una cantidad de veces determinada por sus valores inicial y final.

Ahora vamos a conocer los bucles indefinidos. Estos se diferencian de los otros en que pueden decidir por si mismos si deben repetirse o no. Esta decisión es tomada por el propio bucle en base a una prueba que se realiza en cada ciclo.

Veamos esto con mas detalle:

8.1 Bucle DO UNTIL END

En éste, las instrucciones a repetir se colocan entre los comandos DO y UNTIL. La prueba debe ser colocada entre UNTIL y END:

```
<<
DO instrucciones UNTIL prueba END
>>
```

Lo que hace este bucle es repetir las instrucciones, seguidas por las pruebas, hasta que la prueba deje de ser falsa, o sea, hasta que su resultado sea diferente de cero. Veamos un ejemplo simple:

Escribamos un programa que tome un número de la pila y le reste 2 repetidas veces, hasta que el número sea menor que 0.

```
<< DO 2 – UNTIL DUP 0 < END >>
```

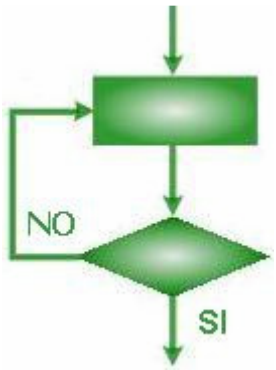


Diagrama reflujo para una Estructura DO...UNTIL...END

Mejor utilicemos un 3 DISP para ver como varia:

```
<< DO 2 – DUP 3 DISP
UNTIL DUP 0 < END >>
```

Veamos otro ejemplo: Escribamos un programa que tome un número de la pila y le sume 1 hasta que el usuario presione CANCEL:

```
<<
DO 1 + DUP 3 DISP UNTIL 0 END
>>.
```

8.2 Bucle WHILE… REPEAT…END

En éste, las instrucciones a repetir se colocan entre los comandos REPEAT y END. La prueba se coloca entre WHILE y REPEAT:

<<

WHILE prueba REPEAT instruc END

>>

Como se puede ver, este bucle realiza la prueba antes de ejecutar las instrucciones. Esta es su diferencia fundamental con el DO UNTIL END.

Otra diferencia importante es que el bucle WHILE ejecuta las instrucciones si el resultado de la prueba es verdadero.

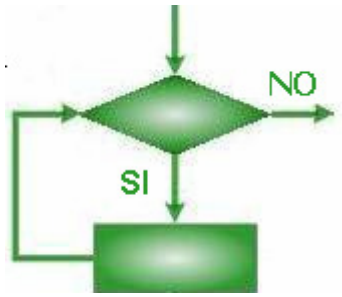


Diagrama de flujo para una Estructura WHILE…REPEAT…END

El bucle DO ejecuta las instrucciones por lo menos una vez, mientras que el WHILE puede que nunca se ejecute.

Escribamos un programa que Tome cualquier cantidad de objetos de la pila y sume todos los que sean números reales y el resto los elimine:

<<

0 WHILE DEPTH OVER TYPE

0 IF ≠ THEN SWAP DROP ELSE + END

END

>>

Ejercicio: Escriba un programa similar al anterior, para instalar librerías.

En el ejemplo anterior hemos colocado una estructura condicional dentro de una estructura de bucle. A esto se le llama **anidar** estructuras. Esta técnica puede ser usada con cualquier tipo de estructura.

CAPÍTULO X. Más Sobre Salida y Entrada de Datos

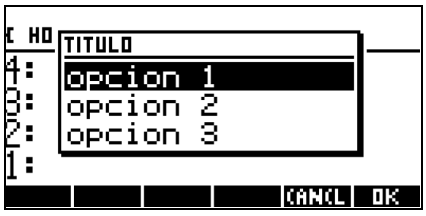
9.1 El menú CHOOSE.

Este menú ofrece una forma elegante de pedir a los usuarios que elijan entre varias opciones.

Veamos el código para un menú de este tipo:

```
<< "TITULO"
{ { "opcion 1" << programa1 >> }
  { "opcion 2" << programa2 >> }
  { "opcion  3" << programa3 >> }
} 1 CHOOSE >>
```

El menú creado se muestra a continuación:



Al elegir una opción, el programa correspondiente es enviado a la pila sin evaluar. También es enviado un 1 que indica que una opción fue elegida.

Si se presiona la tecla ON, y esto se hace cuando se quiere salir del menú sin elegir ninguna opción, simplemente aparecerá un 0 en la pila.

El cero y el uno pueden ser usados para determinar la evaluación o no del programa:

```
<< "TITULO"
{ { "opcion 1" << programa 1 >> }
  { "opcion 2" << programa 2 >> }
  { "opcion  3" << programa 3 >> }
} 1 CHOOSE
<< EVAL >>  IFT
>>
```

Como se puede apreciar, el código para crear un menú CHOOSE es muy simple: Tenemos una cadena de caracteres para el titulo. Luego, las opciones del menú y al final un 1 que indica la posición del cursor al desplegar el menú.

Aunque solo hemos usado tres opciones para el ejemplo anterior, lo cierto es que se puede tener todas las opciones que se necesiten.

Los programas correspondientes a cada opción pueden ser sustituidos por cualquier otro objeto.

Otra cosa que se debe mencionar es que podemos crear menús con opciones que desplieguen otros menús, o sea que los menús CHOOSE se pueden anidar.

9.2 El formulario INFORM.

Este formulario ofrece muchas ventajas a la hora de pedir datos a los usuarios. Para comprender sus grandes ventajas veamos un ejemplo simple:

```
<< "AGENDA"
{ { "NOMBRE" } { "DIREC"} { "TELF"} }
1
{"MARK" NOVAL NOVAL}
{ NOVAL NOVAL  555.5555 }
INFORM >>
```

Al ejecutar el programa tendremos:

Vemos que el campo TELEF contiene una opción por defecto (555.5555). En el código del formulario, la lista anterior al comando INFORM contiene las opciones por defecto. Se pone NOVAL donde no se quiere opción por defecto. Esta lista puede ser sustituida por una lista vacía.

Si se presiona NXT , veremos la opción RESET , la cual, al ser ejecutada, hará aparecer el texto "MARK" en el campo NOMBRE

En el código del formulario, La lista anterior a la lista de valores por defecto contiene los valores de reseteo.

El entero 1 que aparece en el código indica el número de columnas en que se quiere distribuir las opciones. En nuestro caso, una columna.

Este número puede ser sustituido por una lista que de dos números, donde el primero indicara la cantidad de columnas, y el segundo, la separación entre la etiqueta y el campo. Podemos, por ejemplo, sustituirlo por {1 3}

Otra cosa importante es que podemos incluir mensajes de ayuda y restringir el tipo de dato que el usuario podrá introducir en cada campo. Para esto podemos sustituir la lista

```
{ "DIREC"}
```

por esta otra:

```
{ "DIREC"
"PONGA DIRECCIÓN AQUÍ" 2 }
```

El segundo texto es el mensaje de ayuda y el entero 2 indica que en ese campo solo se podrá introducir objetos tipo 2, o sea, cadenas de caracteres.

Si no interesa que haya mensaje de ayuda, podemos sustituir la lista anterior por:

```
{ "DIREC" " "  2 }
```

Como se ha podido apreciar, el INFORM ofrece una gran flexibilidad a los programadores. Además, evita errores que se podrían presentar al usar otros métodos de entrada de datos.

Como desventajas, podemos mencionar las limitaciones de espacio en la pantalla y la poca velocidad.

9.3 Control del Teclado.

La HP48 posee un total de 49 teclas, cada una de las cuales puede ser indicada por un número tres dígitos con el formato: `YX.Z`

La Y nos indica en cual fila del teclado se encuentra la tecla y la X indica la columna.

La tecla MTH, por ejemplo, se encuentra en la fila 2 y columna 1 del teclado.

Finalmente, la Z indica el plano:

PLANO	Z
Sin tecla de cambio	0
Sin tecla de cambio	1
Con tecla de cambio	2
Con tecla de cambio	3
alfabético	4
Alfabético con cambio	5
Alfabético con cambio	6

La tecla MTH, por ejemplo, tiene RAD en el plano 2 y POLAR en el plano 3.

9.3.1 El comando WAIT.

Uno de los comandos más utilizados es el **WAIT.** Este comando puede usarse para detener la ejecución de un programa por un tiempo determinado.

Escribamos un programa que muestre dos mensajes en secuencia, pero que le de al usuario suficiente tiempo para leer ambos textos, digamos 2 segundos para el primero y 3 segundos para segundo:

```
<<
 "PRIMER TEXTO" 3 DISP
2 WAIT
"SEGUNDO TEXTO" 3 DISP
3 WAIT
>>
```

El WAIT también puede detener un programa hasta que el usuario presione una tecla, para esto se debe colocar un cero como tiempo:

EJ.:

```
<< "TEXTO" 3 DISP 0 WAIT >>
```

Al presionar una tecla, se envía a la pila el número **YX.Z** de dicha tecla. Este número pude ser utilizado por una estructura condicional para que el programa decida que hacer en base a la tecla que el usuario presiono.

Ej.:

```
<< 54.1 IF == THEN KILL END >>
```

Podemos utilizar el WAIT para crear programas que tomen decisiones en función de la tecla que se presione. Podemos, por ejemplo, escribir un programa que desplegué un mensaje que diga:

"Presión A para continuar o B para cancelar"

Luego se podría utilizar una estructura condicional para que se ejecute la opción elegida por el usuario.

El siguiente ejemplo presenta un mensaje en la pantalla y espera 2 segundos:

```
<<
 "Programa Ejemplo" 3 DISP 2 WAIT
>>
```

9.4 El comando KEY

Este comando es particularmente interesante. Ayuda en la creación de programas cuyo flujo varíe mientras el programa se ejecuta. Puede resultar especialmente útil para aquellos lectores interesados en crear juego.

Lo que hace el KEY es detectar si una tecla está siendo presionada. En caso afirmativo, devuelve un 1 al nivel **1:** de la pila, y el número de identificación de la tecla al nivel **2:.** (en formato XY)

Si no está siendo presionada ninguna tecla, el KEY devuelve un cero a la pila.

El comando KEY debe ser apoyado por una estructura de bucle ya que realiza su prueba en el momento exacto de ejecución del programa.

Escribamos un programa que tome un número de la pila y lo incremente de uno en uno sucesivamente hasta que se presione la tecla DEL

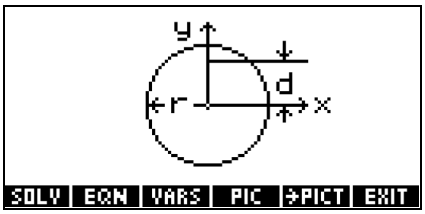
```
<<
CLLCD
DO 1 + DUP 3 DISP
UNTIL KEY DUP
<< SWAP 54 ≠ NOT IFT >> IFT
  END
>>
```

El Análisis del ejemplo anterior queda en manos del lector.

CAPÍTULO XI. Manejo Avanzado de Gráficos

10.1 La utilidad de los dibujos.

Los dibujos pueden ser usados como apoyo para hacer más comprensible la entrada o salida de datos.



Podemos guardar un dibujo como variable y presentarlo al usuario en el momento adecuado (→LCD).

Los dibujos por lo general ocupan mucha memoria, por lo que la opción anterior no siempre es la más conveniente.

Otra opción sería que el propio programa genere su dibujo cuando se ejecute. Para esto existe una serie de comandos, los cuales conoceremos en el presente capítulo.

.

10.2 PICT como variable.

Si tenemos un objeto gráfico en la pila y queremos modificarlo con el editor gráficos, podemos utilizar el programa:

<< PICT STO >>

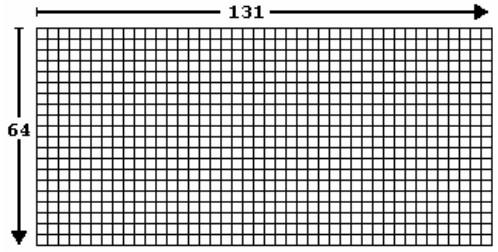
el cual tomara el gráfico de la pila y lo carga en PICTURE. Al entrar al editor gráfico (presione ←) podremos realizar las modificaciones necesarias.

Si queremos mejorar el programa anterior, para que entre automáticamente al entorno PICT, bastara:

<< PICT STO PICTURE >>

Podemos también traer un gráfico desde el editor a la pila con:
<< PICT RCL >>

10.2 Coordenadas.



Un objeto gráfico de tamaño estándar posee 131 puntos de ancho por 64 de alto. Cualquier de sus punto puede ser indicado mediante sus coordenadas, con una lista de la forma:

{ # X # Y }

Esta notación usa como referencia el punto superior izquierdo del gráfico, el cual tiene coordenadas { # 0d # 0d }.

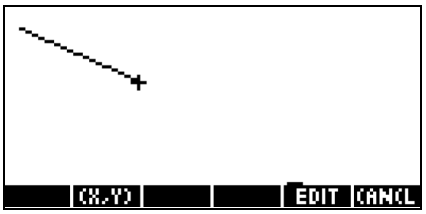
La X indica la distancia horizontal y la Y indica la distancia vertical. El punto inferior derecho de un gráfico estándar tiene coordenadas { # 130d # 63d}

10.8 Comandos para la creación de dibujos.

Podemos crear un dibujo en PICT desde la pila. El comando LINE, por ejemplo, traza una línea entre coordenadas dadas:

<< {# 5 # 7 } { # 41 # 23} LINE >>

Para ver el resultado tenemos que entrar a PICTURE.



Otros comandos de este tipo son los siguientes tipo:

PIXON : Toma una coordenada de la pila y activa el punto localizado en esa coordenada.

PIXOFF : Toma una coordenada de 1: y desactiva el punto localizado en esa coordenada.

PIX? : Sirve para saber si un punto está activado o no. Si está activado devuelve un 1; en caso contrario, un 0.

BOX : Dibuja un recuadro a partir de las coordenadas de sus esquinas.

ARC : Traza un arco centrado en una coordenada (en nivel 4:) con un radio dado (en nivel 3:) y desde un Angulo  (en nivel 2:) hasta otro  (en nivel 1:).

C→PX : Convierte una coordenada de unidades de usuario (X,Y) en una coordenada de punto {#X #Y}.

BLANK : Crea un objeto gráfico en blanco en la pilad de ancho #X (nivel 2:) y alto **#Y** (nivel 1:).

Ej.:

<< # 9d # 6d BLANK >>

.Investigue qué hacen los siguientes comandos:

1) ERASE
2) PX→C
3) TLINE
4) PDIM

10.3 El comando SUB.

Este comando sirve para extraer una sección rectangular de un gráfico. El comando SUB requiere:

3: Gráfico
2: **{ # X1 # Y1 }**
1: **{ # X2 # Y2 }**

La lista del nivel 2: indica la coordenada de la esquina superior izquierda del rectángulo.

La lista del nivel 1: indica la coordenada de la esquina inferior derecha del rectángulo.

El siguiente programa toma un gráfico de la pila y extrae una sección rectangular. Luego muestra dicha sección rectangular en PICT:

```
<<
{ # 10d # 5d} {# 70d # 20d} SUB
PICT STO PICTURE
>>
```

10.4 El comando REPL.

Este comando hace lo opuesto al SUB, es decir que sirve para insertar un gráfico en otro.

El REPL requiere el gráfico receptor, en **3:** ; la coordenada donde se hará la inserción, en **2:** ; y el gráfico a insertar, en **1:.**

10.5 El comando →GROB.

Con este comando puedemos convertir en gráficos objetos como cadenas de texto, nombres, números, listas, programas, etc.

El gráfico puede ser creado tanto con caracteres pequeños como grandes y medianos.

El comando →GROB requiere el objeto en el nivel 2: y un número en 1: que indica el tamaño de caracteres:

```
1 = caracteres pequeños
2 = caracteres medianos
3 = caracteres grandes
0 = gráfico de ecuaciones
```

Veamos algunos ejemplos:

El siguiente programa toma un objeto de la pila y lo convierte en un gráfico con caracteres medianos:

<< 2 →GROB >>

10.6 El comando NEG.

Si colocamos un objeto gráfico en la pila y ejecutamos el comando **NEG** obtendremos un gráfico con color invertido, esto es, los puntos blancos aparecerán negros y los negros aparecerán blancos. Podemos utilizar esto para crear entornos con áreas resaltadas o con una especie de cursor.

```
<<
LCD→
{ # 30 # 16 }
"TRABAJANDO CON GROBS"
1 →GROB
REPL
→LCD  3 FREEZE
 >>
```

Ejercicios:

(a) Investigue sobre los siguientes comandos:

1) GOR
2) GXOR
3) BLANK
4) PVIEW

(b) Escriba un programa que tome doce números de la pila y los organice en una tabla de dos columnas.

(c) Escriba un programa que muestre la siguiente información sobre la calculadora:

1) memoria sin utilizar
2) versión de ROM
3) números de variables en memoria
4) número de librerías instaladas

Para este proyecto necesitara los comandos:

1) MEM
2) VERSION
3) VARS y SIZE

10.6 Creando Animaciones.

La animación es crear el la ilusión de movimiento al presentar una sucesión de objetos gráficos.

La HP48 tiene la capacidad de crear animaciones. Podemos, por ejemplo, hacer ver una pelota rebotando en la pantalla. Para esto bastaría animar una serie de dibujos en los que la pelota esté en diferentes posiciones.

Para la creación de animaciones existe el comando ANIMATE. Para que este comando funcione solo tenemos que llamar a la pila todos los gráficos a animar y luego un número entero que indique su cantidad. Al ejecutar ANIMATE verá como los gráficos son mostrados uno a continuación del otro.

Ejemplo: Si tenemos tres gráficos guardados con los nombre G1, G2 y G3, el siguiente programa los utiliza para crear una animación:

```
<<
G1 G2 G3
3 ANIMATE
>>
```

Ejercicio: Use el método anterior para crear una animación.

10.7 Controlando parámetros de animaciones.

Existe una forma para controlar diversos parámetros de nuestras animaciones. Tale parámetros son:

1) Tiempo entre un gráfico y otro.
2) Número de repeticiones.
3) Posición en pantalla.

Para esto, se debe sustituir el número que indica la cantidad de gráficos a animar, por una lista que contenga los siguientes elementos:

1) Número de gráficos a animar (n)
2) Coordenadas donde se presentara la animación { # X # Y }
3) Tiempo de retardo (t), en segundos.
4) Número de veces que se repetirá la animación (p)

Por tanto, la lista debe tener la forma siguiente:

```
{ n { # X # Y } t  p }
```

En el ejemplo de la sección 10.6, podemos sustituir el **3** por la siguiente lista **:**

```
{ 3 {#1d  #18d} 1 5}
```

CAPÍTULO XII. Representaciones Gráficas Bidimensionales

La HP48 dispone de un conjunto de comandos para representaciones gráficas. En esta sección se mostrará cómo hacer programas para graficar funciones con coordenadas cartesianas.

13.1 Funciones.

Lo primero es seleccionar el tipo de representación gráfica con el comando FUNTION El comando **DRAW** sirve para crear representaciones gráficas. Este comando requiere la variable EQ que contendrá la función a ser graficada.

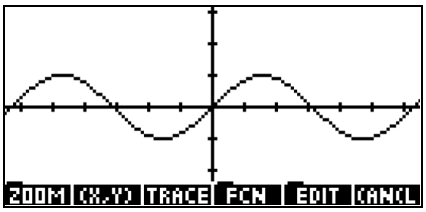
Existe además el comando **STEQ**, el cual toma la función de la pila y la almacena en EQ.

El comando **DRAX** traza los ejes cartesianos.

Ej.:

Escribamos un programa que construya el gráfico de la función **y = sin (x**). El código será:

```
<<
{ PICT  PPAR }  PURGE
'Y=SIN(X)' STEQ
RAD  DRAX DRAW
PICTURE
>>
```



Lo primero que hace el programa es una limpieza PICT.

Junto a PICT, se elimina también la variable PPAR. Esto hace que el gráfico se genere con los parámetros por defecto (ver página 22-14 de la *Guía de Usuario*.)

Luego se almacena la función de X en la variable EQ (**STEQ**), se fija el modo Radianes (**RAD**) y se trazan los ejes (**DRAX**)

Finalmente se traza el gráfico (**DRAW**) y pasamos a PICT para ver el resultado (**PICTURE**)

El programa anterior pudo también escribirse como:

```
<<
{ PICT  PPAR }  PURGE
'SIN(X)' STEQ
RAD  DRAX DRAW
PICTURE
>>
```

o' como:

```
<<
{ PICT  PPAR }  PURGE
<< X SIN >> STEQ
RAD  DRAX DRAW
PICTURE
>>
```

13.2 Coordenadas polares.

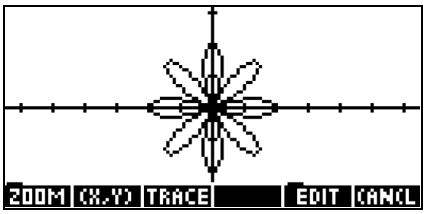
Veamos cómo graficar funciones descritas en coordenadas polares.

Para elegir este modo de representación gráfica ejecutamos el comando POLAR.

Debemos especificar las variables independiente y dependiente. Esto lo hacemos con los comandos INDEP y DEPND.

Veamos un programa que genere el gráfico para la ecuación $R = 2\cos(4\theta)$

```
<<
{ PICT PPAR } PURGE
RAD
POLAR
R DEPND
θ INDEP
'R=2*COS(4*θ)' STEQ
DRAX DRAW PICTURE
>>
```



El ejemplo anterior también pudo haberse escrito como:

```
<<
{ PICT PPAR } PURGE
RAD
POLAR
θ INDEP
'2*COS(4*θ)' STEQ
DRAX DRAW PICTURE
>>
```

o' como:

```
<<
{ PICT PPAR } PURGE
RAD
POLAR
R DEPND
θ INDEP
<< θ 4 * COS 2 * >> STEQ
DRAX DRAW PICTURE
>>
```

13.3 Varios gráficos en sucesión.

Si queremos que el programa construya el gráfico para varias ecuaciones solo tenemos que introducir dichas ecuaciones en una lista.

```
<<
{ PICT  PPAR } PURGE
{ 'SIN(X)'  'COS(X)' }  STEQ
RAD  DRAX DRAW
PICTURE
>>
```

Si el indicador –28 está activado, los gráficos se construirán en sucesión. En caso contrario se construirán juntos.

Veamos otro ejemplo:

```
<<
{ PICT  PPAR } PURGE
{ '3*Y=X^2 - X'  'X-1'  'LN(X)' } STEQ
RAD  DRAX DRAW
PICTURE
>>
```

13.4 Otros comandos útiles.

RES : Toma un número de la pila y lo usa para fija el espaciado de los puntos de la representación gráfica.

XRNG : Toma dos números de la pila y los usa para especificar el rango de visualización del eje horizontal.

YRNG : Toma dos números de la pila y los usa para especificar el rango de visualización del eje vertical

CENTR : Fija el centro de visualización de una representación gráfica en las coordenadas dadas en 1:.

SCALE : Toma dos número de la pila y los usa para fijar la escala horizontal y vertical de los ejes.

***W** : Multiplica la escala por un número dado en 1:.

***H** : Multiplica la escala vertical por un número dado en 1:.

ERASE : Borra PICT.

AXES : Se usa para trasladar los ejes. Requiere la coordenada (x,y) del nuevo punto de intersección en 1:.

<< ... (1,2) AXES DRAX ... >>

AXES : Puede usarse también para etiquetar los ejes. Requiere la lista con las etiquetas

```
<< …
 { "h-etiqueta" "v-etiqueta"}
AXES
LABEL  ...
>>
```

13.5 Ecuaciones Parametricas.

En algunos casos, las coordenadas X y Y se pueden expresar como ecuaciones de una misma variable independiente que se denomina *Parámetro*.

$$X = F(t) \quad Y = G(t)$$

En la HP48 esto también puede graficar en esta forma.

Escribamos un programa que construya el gráfico en que:

$$F(t) = 2t \quad y \quad G(t) = 3t\text{^}2$$

```
<<
{ PICT  PPAR } PURGE
RAD
PARAMETRIC
{ t  0  6 }  INDEP
'2*t + i*3*t^2'  STEQ
DRAX  DRAW  PICTURE
>>
```

Análisis:

La variable independiente se introduce como una lista que contiene los siguientes tres datos:

t: Nombre de la variable independiente.

0: Menor valor de la variable independiente que se desea representar (LO)

6: Mayor valor de la variable independiente que se desea representar (HI)

Las ecuación fue introducida en formato:

```
'F(t) + G(t)*i'
```

¿Y si queremos crear simultáneamente varios gráficos? Simplemente hay que usar una lista de ecuaciones en lugar de una ecuación:

Ej.:

```
{ 'F1(t) + G1(t)*i'
  'F2(t) + G2(t)*i'
  'F3(t) + G3(t)*i' }
```

En general, EQ acepta ecuaciones, expresiones o programas, siempre que al ser evaluados devuelvan un número complejo.

13.6 Cónicas.

Para este tipo de representaciones gráficas usamos el comando CONIC. El programa siguiente construye una Elipse.

```
<<
      { PICT  PPAR } PURGE
      CONIC
      0.1  0.1   0.1   RES
      { X  -2  2 }  INDEP
      '5*X^2+9*Y^2-17'  STEQ
      DRAX  DRAW  PICTURE
>>
```

Análisis:

El gráfico es trazado en dos ramas. Si el indicador –1 está activado, solo se mostrará la rama principal. Desactive el indicador –1 para que aparezca la sección cónica completa.

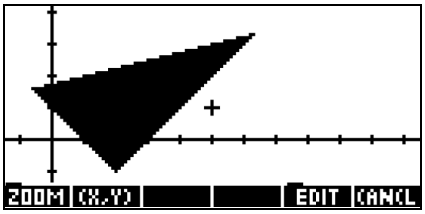
Obsérvese que se ha reducido el intervalo entre los puntos (0.1 RES) esto es necesario para evitar las discontinuidades visuales que se producen por el trazado en dos ramas.

13.7 Graficando desigualdades.

Para esto tenemos el tipo de representación gráfica THRUTH.

El siguiente programa muestra gráficamente los puntos que satisfacen el sistema de desigualdades: X+Y≥1, 4X≤X+7 y Y≥X-3.

```
<< RAD
{ PICT PPAR} PURGE
TRUTH
'X+Y≥1 AND 4X≤X+7 AND Y≥X-3'
STEQ
(5,1) CENTR DRAX DROW PICTURE
>>
```



Obsérvese que el programa fija el centro de visualización en el punto (5,1).

CAPÍTULO XIII. Asuntos Diversos

12.1 Programando los modos.

Existe una serie de comandos que sirven para controlar los modo de funcionamiento de la HP48:

RAD : Fija el modo angular en Radianes

DEG : Fija el modo angular en grados sexagesimales.

GRAD : Fija el modo angular en grados centesimales.

RECT : Fija el modo de coordenadas rectangulares.

CYLIN : Fija el modo de coordenadas cilíndricas.

SPHERE : Fija el modo de coordenadas esféricas.

STD : Fija el modo de pantalla estándar.

FIX : Toma un número Entero **n** de la pila y activa el modo FIX con n lugares decimales.

SCI : Toma un número Entero **n** de la pila y activa el modo de pantalla científico con n lugares decimales.

ENG : Toma un número **n** de la pila y activa el modo de pantalla en ingeniería, con **n+1** dígitos significativos.

12.2 Otros comandos de Pila.

En capítulos anteriores vimos los comandos siguientes:

DUP : Duplica el objeto del nivel 1:

DUP2 : Duplica los objetos de los niveles 1 y 2:

SWAP : Invierte los niveles 1: y 2:

DROP : Elimina el objeto del nivel 1:

DROP2 : Elimina los objetos de los niveles 1: y 2:

CLEAR : Elimina todos los objetos de la pila.

ROT : Mueve el objeto del nivel tres al nivel 1:

DEPTH : Devuelve el número de objetos que hay en la pila.

OVER : Devuelve una copia del objeto que está en el nivel 2:

A continuación presentamos el resto de los comandos para manipulación de pila:

PICK : Devuelve una copia del objeto que está en el nivel n+1

Ej.:

4:	**a**		4:	**a**
3:	b	→	3:	b
2:	c		2:	c
1:	3		1:	**a**

DUPN : Duplica n objetos, comenzando en el nivel 2:, n se da en el nivel 1: de la pila.

Ej.:

6:			6:	a
5**:**			5:	b
4:	a	→	4:	c
3:	b		3:	a
2:	c		2:	b
1:	**3**		1**:**	**c**

DROPN : Elimina n objetos de la pila, comenzando en el nivel 2:

Ej.:

5**:**	a		5:	
4:	**b**		4:	
3:	**c**	→	3:	
2:	**d**		2:	
1:	3		1**:**	a

ROLL : Mueve el objeto en el nivel n+1 al nivel 1:.

Ej.:

5**:**	**a**		5:	
4:	b		4:	b
3:	c	→	3:	c
2:	d		2:	d
1:	4		1**:**	**a**

ROLLD : Mueve el objeto del 2: al nivel n+1.

Ej.:

5**:**	a		5:	
4:	b		4:	**d**
3:	c	→	3:	a
2:	**d**		2:	b
1:	4		1**:**	**c**

12.3 Manejo de Memoria.

En capítulos anteriores vimos los comandos MEM, PURGE y VARS, los cuales pueden usarse para programas relacionados con la memoria. Otros comandos de este tipo son:

BYTES : Devuelve el tamaño (en bytes) del objetos del nivel 1.

CLVAR : Borra todas las variables de usuario.

CLUSR : Borra todas las variables de usuario.

LASTARG : Devuelve el o los argumentos del último comando ejecutado, para que puedan ser utilizados de nuevo.

Ejemplo: Utilicemos el LASTARG para un programa que tome dos números de la pila y nos devuelva su suma, resta y multiplicación:

```
<< + LASTARG  - LASTARG * >>
```

CRDIR : Sirve para crear directorios. Requiere un nombre para el directorio en 1:.
Ejemplo: El siguiente programa crea un directorio con el nombre DIRX:

<< 'DIRX' CRDIR >>

PGDIR : Elimina un directorio y todo su contenido. Solo requiere el nombre del directorio en la pila.

UPDIR : Convierte el directorio superior en directorio actual.

MERGE1 : Fusiona una tarjeta RAM con la memoria de usuario. La tarjeta debe estar instalada en la ranura 1.

FREE1 : Hace lo contrario al comando anterior. O sea que libera una tarjeta RAM en la ranura 1 que está fusionada con la memoria de usuario.

Si queremos que al liberar la tarjeta ésta contenga algunos de los objetos en memora, solo tenemos que poner en la pila una lista con los nombres de dichos objetos, antes de ejecutar FREE1.

Investigue qué hacen los comandos: STO+, STO-, STO* y STO/

12.4. Trabajando con librerías.

LIBS: Devuelve una lista de todas las bibliotecas del directorio actual.

ATTACH: Añade una librería al directorio actual. Requiere en 1: el identificador **:puerta:número** de la librería.

DETACH : Separa una librería de un directorio. Para esto, primero se debe entrar al directorio donde está incorporada la librería. Luego hay que colocar el número de la biblioteca en la pila y finalmente ejecutar DETACH

Escribamos un programa para eliminar librerías. Este programa solo requerirá el identificador **:puerta:número** de la librería que se quiere eliminar. El código es:

<< DETACH LASTARG PURGE >>

12.5 La Biblioteca de Constantes.

La HP48 posee un archivo con 39 cantidades y constantes físicas

Se puede entrar a la biblioteca de Constantes ejecutando el comando **CONLIB**

También podemos recuperar una constante en la pila colocando su símbolo en 1: y ejecutando el comando **CONST**

El siguiente programa recupera en la pila el valor de la aceleración de la gravedad:

<< g CONST >>

12.6 La Biblioteca de Ecuaciones.

La HP48 posee una biblioteca con más de 300 ecuaciones que permiten resolver gran cantidad de problemas técnicos y científicos sencillos.

Podemos entrar a la biblioteca de ecuaciones ejecutando el comando **EQNLIB**

Esta biblioteca consta de 15 temas:

N	TEMA
1	Columnas y Vigas
2	Electricidad
3	Fluidos
4	Fuerzas y Energía
5	Gases
6	Transmisión de Calor
7	Magnetismo
8	Movimiento
9	Óptica
10	Oscilaciones
11	Geometría Plana
12	Geometría Sólida
13	Elementos de Estado Sólido
14	Análisis de Esfuerzos
15	Ondas

Cada uno de estos temas se divide a su vez en subtemas. El tema de geometría plana, por ejemplo, se divide en cuatro subtemas:

n	Subtemas
1	Cono
2	Cilindro
3	Paralelepípedo
4	Esfera

Algunos de los subtemas incluyen un diagrama de acompañamiento

En el **Apéndice F** de la *Guía de Usuario*, se encuentra una lista completa los Temas y Subtemas de la Biblioteca de Ecuaciones.

12.6.1 Comando SOLVEQN.

Activa el entorno para resolución de problemas con la biblioteca de ecuaciones incorporada.

Este comando requiere un número **N (**en nivel 3:) que representa el Tema; un número **n** (en nivel 2:) que representa el subtema**;** y un tercer número *n* (en nivel 1:) que puede ser **0** o **1** .

Si *n*=1, se cargara en PICT el diagrama que acompaña al subtema. Podrás verlo al entra al Editor Gráfico.

Ej.: El siguiente programa activa el modo de resolución para problemas de caída libre:

<< 8 2 0 SOLVEQN >>

12.7 Menús.

Los cientos de comandos y funciones incorporados en la HP48 se encuentran organizados en menús.

Hay un número único asociado a cada menú. El menú VAR, por ejemplo, es el número 2 y el MTH es el número 3.

En el A**péndice C** de la *Guía de Usuario* se presenta una lista de los menús y sus números correspondientes.

Si estamos dentro de un menú y queremos saber su número, solo debemos ejecutar el comando **RCLMENU**. Como resultado obtendrás un número de la forma: **X.Y,** donde X es el número del menú y Y indica la pagina de éste en que nos encontramos.

Podemos entrar a cualquier menú, desde cualquier sitio, si escribimos su número y ejecutamos el comando **MENU**.

Ej.:

Escribamos un programa que, al ejecutarse, tome un programa de la pila, lo cargue en el Debuger y nos muestre el menú para depuración.

```
<< DBUG 41 MENU >>
```

También podemos hacer programas para crear menús personalizados, para esto debemos usar **CST.** (ver Cáp. 30 de la *Guía de Usuario*)

Ej.:

El siguiente programa crea un menú personalizado para la resolución de problemas de Mecánica de Fluidos:

```
<<
{ DARCY
{ "PRES" << 3 1 0 SOLVEQN >> }
{ "BERN" << 3 2 0 SOLVEQN >> }
{ "FWL" << 3 3 0 SOLVEQN >> }
{ "FIFP" << 3 4 0 SOLVEQN >> }
{ "PC" << 52 MENU >> }
{ "TC" << 53 MENU >> }
{ "VC" << 58 MENU >> }
{ "PSTD" << StdP CONST >>}
{ "TSTD" << StdT CONST >>}}
'CST' STO
1 MENU >>
```

12.8 Trabajando con Fechas y Horas.

La HP48 dispone de una gran cantidad de comandos para trabajar con fechas y horas.

El formato para la fecha es controlado por el indicador 42. Si el indicador 42 está activado la fecha se presentara en formato **dd.mmaa** (día, mes y año) ; en caso contrario la fecha aparecerá en formato **mm.ddaa** (mes, día y año)

El formato para la hora es controlado por el indicador 41. Si está desactivado tendremos formato de 12 horas, pero si está activado tendremos formato de 24 horas.

El que se muestre el reloj o no va a depender del estado del indicador 40

Veamos algunos comando relacionados con la fecha y la hora:

DATE : coloca la fecha actual en la pila. El formato dependerá del estado del indicador - 42.

DATE+ : Toma un número de días del nivel 1: y lo suma a una fecha dada en 2:. El siguiente programa suma 37 dias a la fecha actual.:

<< DATE 37 DATE+ >>

DDAYS : determina el número de días existentes entre dos fechas dadas.

TIME : coloca la hora actual en la pila en formato HMS (horas, minutos y segundos)

HMS+ : toma dos números de horas de la pila (en formato HMS) y las suma

HMS- : resta dos números de horas en formato HMS.

La HP también maneja el tiempo en formato decimal. Una hora y treinta minutos, por ejemplo, que es una hora y media, puede expresarse en HMS como:

1.300000.

pero, en formato decimal sería:

1.5

→**HMS** : convierte de decimal a HMS.

HMS→ : convierte de HMS a decimal.

TSTR : toma una fecha de 2: y una hora de 1: y devuelve una secuencia de texto.

Prueba el siguiente programa:

<< DATE TIME TSTR MSGBOX >>

La HP considera no validas las fechas anteriores al 15 de Octubre de 1582 o' posterior al 31 de Diciembre del 9999.

→**DATE** : Fija la fecha del sistema en la fecha especificada en 1:.

Nota: Se aceptan fechas entre el 1 de enero de 1991 y 31 de Diciembre de 2091.

12.9 Alarma.

ACK : Reconoce la última alarma mostrada (ver pagina 26-4 de la *Guía de Usuario*)

ACKALL : Reconoce todas las alarmas producidas.

El indicador -44 controla el almacenamiento de las alarmas repetitivas que se reconocen.

RCLALARM : De la lista de alarmas del sistema, recupera una alarma especificada por un número en 1:.

Formato:

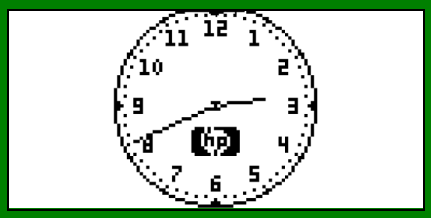
{ fecha hora mensaje repeticiones }

El número de repeticiones se mide en tic tac (1 tictac=1/8192 segundo).

STOALARM : Toma una alarma del nivel 1 y la almacena en la lista de alarmas del sistema.

Retos:

(a) Escriba un programa que muestre la hora como un reloj análogo:



(b) Escriba un programa que muestre el calendario del mes y que actualice el día y la hora cada vez que se ejecute:

(c) Escriba un programa que mida el tiempo que toma otro programa para ejecutarse.

12.10 Estadística.

12.10.1 Comandos diversos.

STOΣ : Toma una matriz del nivel 1: y la almacena en ΣDAT.

CLΣ : Elimina los datos estadísticos de ΣDAT

MEAN : Calcula la media de los datos estadísticos de ΣDAT.

SDEV : Calcula la desviación estándar de cada una de las columnas de ΣDAT**.**

VAR : Calcula la variación de las columnas de ΣDAT.

PSDEV : Calcula la desviación estándar poblacional.

PVAR : Calcula la varianza poblacional.

NΣ : Devuelve el número de datos de ΣDAT.

XCOL : Toma un Entero n de la pila y selecciona la n-sima columna de ΣDAT como la de variable independiente.

YCOL : Toma un Entero n de la pila y selecciona la n-sima columna de ΣDAT como la de variable dependiente.

ΣX : Devuelve la suma de los datos de una columna independiente de ΣDAT.

ΣY : Devuelve la suma de los datos de una columna dependiente de ΣDAT.

ΣX^2 : Devuelve la suma de los cuadrados de una columna independiente de ΣDAT.

ΣY^2 : Devuelve la suma de los cuadrados de los datos que se encuentran en una columna dependiente de**ΣDAT.**

ΣX*Y : Devuelve la suma de los productos de los datos que se encuentran en las columnas independientes y dependientes de ΣDAT.

MINΣ : Devuelve los valores mínimos de cada columna.

MAXΣ : Devuelve los valores máximos de cada columna.

TOT : Devuelve las sumas de las columnas de la matriz de ΣDAT.

Σ+ : Agrega una fila a la matriz ΣDAT. Requiere el vector fila en 1:.

Σ- : Extrae un vector fila de ΣDAT. No se requiere ningún argumento en la pila.

12.10.2 Ajuste de curvas.

LINFIT : Fija el modo de ajuste de curvas en lineal.

LOGFIT : Fija el modo para ajuste de curvas en logarítmico.

EXPFIT : Fija el modo para ajuste de curvas en exponencial**.**

PWRFIT : Fija el modo para ajuste de curvas en potencias.

BESTFIT : Selecciona el modelo de estadísticas produciendo el mayor coeficiente de correlación (valor absoluto) y ejecuta LR.

ΣLINE : Devuelve la línea más adecuada para los datos de ΣDAT de acuerdo con el modelo de estadísticas seleccionado.

LR : Calcula la regresión lineal.

12.10.3 Pronósticos.

PREDX : Devuelve el valor pronosticado para una variable independiente dado el valor de la variable dependiente en 1:.

PREDY : Devuelve el valor pronosticado para una variable dependiente dado el valor de la variable independiente en 1:.

CORR : Calcula el coeficiente de correlación de los datos estadísticos de ΣDAT.

COV : Calcula la covariacion de los datos estadísticos de ΣDAT.

PCOV : Calcula la covariacion poblacional.

12.10.4 Probabilidad.

COMB : Toma un número **n** (del nivel 2) y otro **m** del nivel 1: y calcula el número de combinaciones de **n** elementos tomados de **m** en **m**.

PERM : Toma un número n del nivel 2: y otro m del nivel 1: y calcula el número de permutaciones de n elementos tomados de m en m.

RAND : Devuelve un número aleatorio **n** (**0≤n≤1**) Cada número aleatorio se convierte en la semilla del siguiente.

RDZ : Toma un número real del nivel 1: y lo utiliza como semilla para el siguiente número aleatorio. Si encuentra un 0 en el nivel 1:, se creara una semilla basada en la hora del reloj.

! : Toma un número **n** de la pila y si es entero devuelve su producto factorial. Para números no enteros devolverá **Γ(n+1)**

12.10.5 Distribuciones estadísticas.
Ver pagina 12-5 de la *Guía de Usuario*.

12.11 Librerías

Al escribir un programa largo es mucho más cómodo dividirlo en pequeñas subrutinas. No hay que ser un experto para saber esto. Pero, esto puede traer confusión al usuario, ya que el menú aparece repleto de variables.

Puede ocurrir, por ejemplo, que el usuario ejecute una variable equivocada. También puede ocurrir que el usuario borre o altere alguna de las subrutinas y corrompa el programa.

Las librerías son un medio para proteger nuestros programas de todos los problemas anteriores ya que permiten tener cualquier cantidad de subrutinas y solo mostrar al usuario los comandos necesarios.

Otra ventaja de las librerías es que sus programas pueden ser invocados desde cualquier directorio ya que operan de manera muy similar a los propios comandos de la calculadora.

Cómo convertir nuestros programas en librerías.

Podemos convertir programas en librerías mediante la utilización de programas especiales que toman el directorio donde tienes el programa y lo convierten en librería. Para hacerlo desde la misma calculadora existe por ejemplo el programa →**LIB**← de Detlef Mueller. Para más información vea la documentación adjunta a dichos programas.

12.12 El comando SYSEVAL.

Estamos frente a uno de los más peligrosos y potentes comandos del lenguaje User Rpl, por lo que sugiero que ponga su HP a un lado y no la toque hasta leer completamente esta sección.

Para comprender el funcionamiento de este comando se debe saber que los comandos de la HP48 se encuentran almacenados en su memoria ROM, en una dirección definida por un número hexadecimal de cinco dígitos:

En la ROM R, el comando DUP se encuentra en la dirección # 1FB87h y el comando ABS en la dirección #0003Dh.

Para saber la versión de ROM que posee su HP48, ejecute el comando VERSION.

Lo que hace el SYSEVAL es tomar una dirección de ROM (del nivel 1:) y evaluar la instrucción que se encuentra en dicha dirección.

Pero, ni se le ocurra ponerse a evaluar direcciones sin estar seguro de que la ROM de su HP es la correspondiente, ya que estas direcciones pueden variar de una ROM a otra, y el evaluar una dirección equivocada puede llegar a producir daños en la memoria.

.

Usted se estará preguntando qué razones tendríamos para ejecutar un comando con SYSEVAL si solo bastaría escribir el comando. Lo que sucede es que los programadores en User Rpl no tiene acceso a todos lo comandos disponibles

En el lenguaje System RPL, los programadores tienen acceso mayor cantidad de comandos que en el User Rpl. Esta es una de las razones de que sea más poderoso.

En el User Rpl, los comandos revisan la pila antes de ejecutarse y si no encuentran los datos que necesitan producen un mensaje de error y detiene su ejecución. Esto evita que se produzcan errores internos que pudieran ocasionar situaciones no deseadas.

En el System Rpl casi nunca se hace verificación de pila y el programador debe asegurarse de proporcionar a cada comando los datos que requiere. Esto hace que los programas en System sean más rápidos ya que en el User se hacen con frecuencia chequeos que no son necesarios.

El comando SYSEVAL nos permite ejecutar comandos del System Rpl, pero si al hacer esto el comando no encuentra en la pila el argumento que necesita, se puede producir errores.

12.13 El comando LIBEVAL.

Este comando es muy parecido al SYSEVAL, solo que en lugar de evaluar una dirección de memoria, evalúa un número XLIB.

Es importante decir que el LIBEVAL es también muy peligroso y se debe tener mucho cuidado al usarlo.

EL LIBEVAL toma un entero binario de la forma #LLLFFFh , donde LLL es el ID de la librería y FFF es el número de tres dígitos de la función.

En la siguiente tabla se muestran algunas XLIB útiles.

Función LIBEVAL	
La ventana de CMD	# B2000h
Entrada a Chars	# B2001h
Menú de MODOS	# B41C1h
Menú de los Indicadores.	# B41CFh
Entrada directa a MEMORY	# B41D7h
Entrada directa a SOLVE	# B4000h
Menú de Solve Equation	# B4001h
Menú de Solve difeq Equation	# B4017h
Menú de Solve polynomial	# B402Ch
Menú de sistema de ecuaciones	# B4033h
Menú de TVM	# B4038h
Menú de PLOT	# B4045h
Menú de SYMBOLIC	# B4113h
Menú de Integrales	# B4114h
Menú de diferenciales	# B4122h
Menú de TAYLOR	# B412Bh
Menú de ISOLATE VAR	# B412Dh
Menú de ecuación Cuadratica	# B4130h
Menú de manip. de expresiones	# B4131h
Menú de la Aplicación TIME	# B4137h
Entrada de las alarmas	# B4138h
Set time and Date	# B415Bh
Catalogo de alarmas	# B416Eh
Menú de STAT	# B4175h
Menú de Estad. De una variable	# B4176h
Menú de Frecuencias	# B417Dh
Menú de ajustes de datos	# 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	# B41A8h
Menú de recepción de HP-HP	# B50FFh