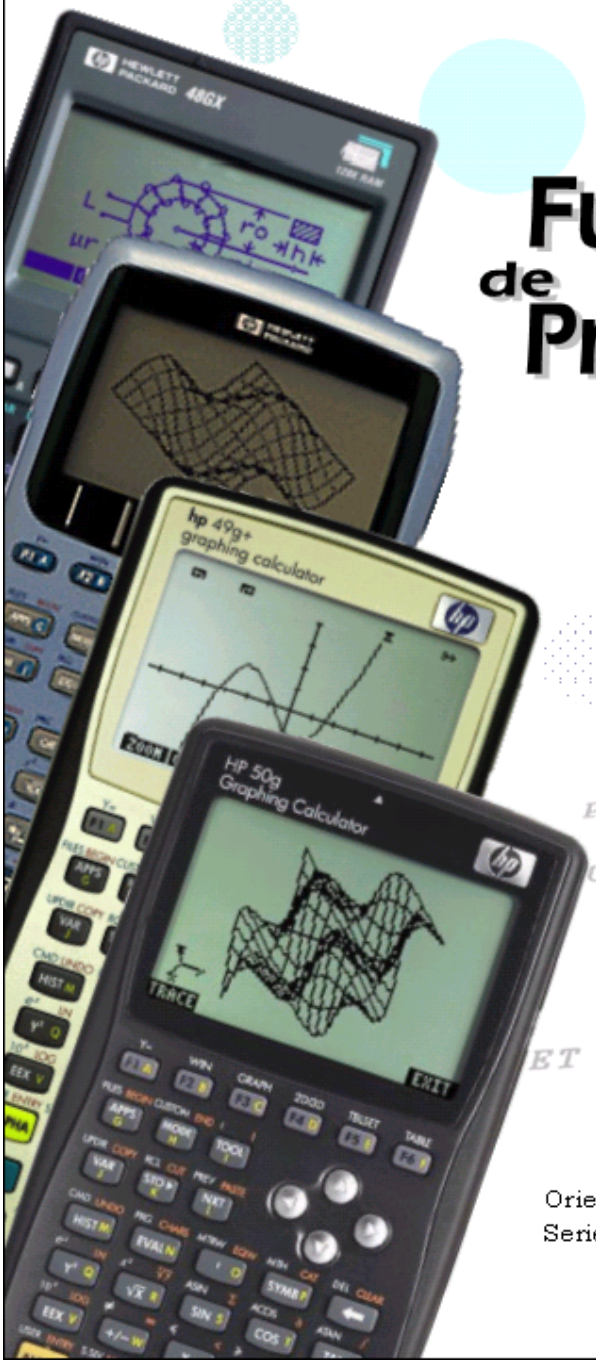
UNIVERSIDAD TECNICA DE ORURO
FACULTAD NACIONAL DE INGENIERIA
CARRERA DE INGENIERIA CIVIL



Fundamentos de Programación en User RPL

calculadoras HP Series 48/49/50

Sandro Choque Martínez

Manual de programación
Orientado en calculadoras HEWLETT PACKARD
Series: 48G,48G+,48GX,49G,48GII, 49G+ y 50G

Fundamentos de Programación en User RPL

Orientado en calculadoras HEWLETT PACKARD
Series: 48G, 48G+, 48GX, 49G, 48GII, 49G+ y 50G

Sandro Choque Martínez

TERCERA EDICION

LA PRESENTACION Y DISPOSICION EN CONJUNTO DE:

FUNDAMENTOS DE PROGRAMACION EN USER RPL ORIENTADO EN CALCULADORAS HEWLETT PACKARD SERIES 48/49/50

CARRERA DE INGENIERIA CIVIL
Pagador No. 6548, Ballivián y San Felipe
Tel.-Fax: (591-2) 5280585

TERCERA ECICION

Realizado en Oruro, Bolivia

El autor <http://sandrusfni.googlepages.com/about.html>

About of sandrus

:: Personal data ::	
Name:	Sandro Choque Martínez (also known as sandrus)
Nationality:	Bolivian
Education:	Civil Engineering UTO-FNI
Laguage:	Spanish and Quechua
Phone:	591-25246350
Email:	sandrus@bolivia.com
:: technical specialties ::	
Computers Experience:	System Windows, Programming languages C, C++, Java, Pascal, Delphi, Visual Basic
Development Web:	Languages Java Script HTML, ASP, PHP, VBasic, .NET, C#
Soil mechanics:	Laboratory Practice, Computational
Polymers:	Works with glass fiber, carbon
:: Other activities ::	

Capítulo I. Conceptos básicos

Capítulo II. Resolución de problemas con la calculadora

Capítulo III. Fundamentos de Programación

Capítulo IV. Estructuras de Programación

Capítulo V. Ejercicios de aplicación

Apendices

El capitulo V y los apendices, solo estan disponibles para la versión impresa.

Para adquirir la versión impresa del libro Completo + 1 CD con más de 3500 programas y documentos solo tienes que contactarte con el autor.

Prólogo

Este versátil libro está dirigido en primer lugar a los alumnos que inician sus estudios en la Facultad Nacional de Ingeniería de la Universidad Técnica de Oruro. Para todos aquellos afortunados que cuentan con una calculadora HP48/49/50 y a todas aquellas personas que les interesa el mundo de la programación. En este, se ha puesto énfasis especial en el diseño como texto para cursos introductorios en las materias de Ciencias de la Computación y Metodología de la Programación o como suplemento para cursos intermedios o avanzados con el uso de la calculadora. Este texto puede ser útil a un público más amplio, que incluye a alumnos de cursos superiores de la FNI, a Ingenieros y a profesores que quieran conocer más de cerca las posibilidades que tiene de trabajar con una calculadora tan maravillosa como es la HP48/49/50.

Se ha pretendido llegar a un equilibrio entre el detalle de las explicaciones, la amplitud de temas tratados y el número de páginas. En algunos casos, junto con las instrucciones introducidas por el usuario, se incluye la salida de pantalla de nuestra calculadora; en otros casos no se incluye dicha salida, pero se espera que el lector disponga de una calculadora y vaya introduciendo las instrucciones a la vez que avanza en estas páginas.

También se anima al lector a ampliar los temas del texto, con la ayuda en la World Web Wide *(WWW)* donde encontrarás toda la documentación del libro en formato electrónico (e-book) en http://sandrusfni.googlepages.com/ que está disponible online (En línea) a través de Internet. En cualquier caso recuérdese que la informática moderna, más que en saber consiste en saber encontrar en pocos segundos lo que se necesita, tal motivo le ahorramos su tiempo, brindándole recursos para descargar con material de apoyo y herramientas que le servirán en sus estudios en la Facultad. También se adicionan algunos enlaces, donde podrás encontrar el material que necesitas para la resolución de problemas en Ingeniería, para estudiantes y profesionales.

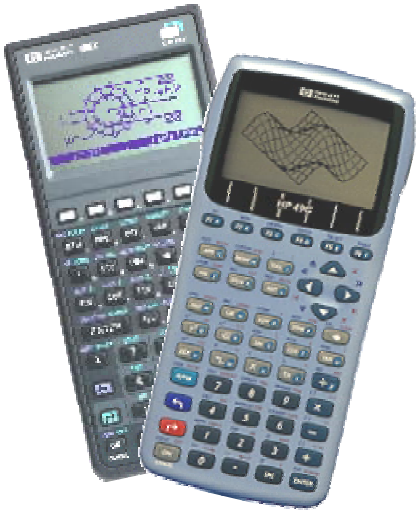
Aprecio enormemente el apoyo de mis queridos compañeros de la facultad, recurso humano que me apoyo anímicamente en la elaboración de este texto para todos nuestros amigos chivatos como un aporte más a la gloriosa Facultad Nacional de Ingeniería.

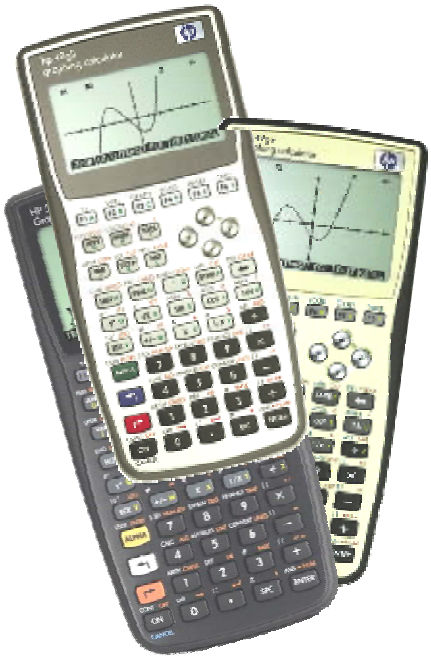
Sandro Choque Martínez

Capítulo I

Conceptos básicos

uladoras Hewlett Packard 48/49/50 demostrado, desde su aparición ser una mienta significativamente útil tanto a nivel mico como a nivel profesional, y en especial campo de la Ingeniería y otras ciencias s. Cabe preguntarse acerca de cómo ha o durante tanto tiempo sin ser superada modelo propio ni de la competencia en e tiempo. La clave de su éxito no radica en spectos diferenciadores respecto de otros elos como: la versatilidad de sus aplicaciones, memoria, albergando librerías y s incorporadas que es efectivamente un mputador simbólico y numérico que facilita el análisis matemático de problemas en una variedad de disciplinas desde matemáticas mentales hasta temas avanzados de ciencia e

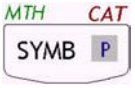


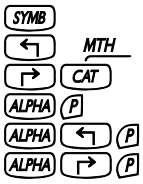


La capacidad de programación de la calculadora, hasta el más bajo nivel permite desarrollar programas eficientes para propósitos específicos, ya sean para aplicaciones matemáticas avanzadas, solución a problemas específicos, o colección de datos; los lenguajes de programación disponibles en la calculadora la convierten en un equipo computacional muy versátil. Cualquier evento realizado por las funciones implementadas puede ser manipulado e incluido en programas propios creados por el usuario. A este efecto implementa un lenguaje propio: el User-RPL. Todas las instrucciones, incluso las más internas son accesibles igualmente, mediante otro lenguaje implementado: el System-RPL. Éste último ha permitido que las funciones incluidas inicialmente por la calculadora queden sobradamente superadas por aquellas creadas por los programadores. Esta característica única de programabilidad absoluta ha dado lugar a la aparición de programas que han extendido la aplicabilidad de la calculadora como: juegos, reproductor de música, emisor, receptor de infrarrojos para equipos domésticos, reproductor de animaciones gráficas, alarma, etc.

1. Manejo de teclado

El teclado de la HP tiene tres a seis niveles de funciones, cada uno de los cuales contiene un conjunto diferente de teclas. Por ejemplo la tecla [SYMB] tiene las seis funciones.





Teclado primario: muestra el menú simbólico
Cambio izquierdo: muestra el menú MATH (Matemáticas)
Cambio derecho: muestra el menú CAT (Catalogo)
Teclado alfabético: escribe la letra mayúscula P
Teclado alfabético cambio izq.: escribe la letra minúscula p
Teclado alfabético cambio der.: escribe el símbolo π

De similar forma son en las series HP 48G/49G, haciendo notar el color y la posición de las etiquetas en la tecla, para las teclas de cambio. Para cancelar una tecla de cambio, pulsar de nuevo la misma. Para cambiar a la otra tecla de cambio, pulsar la otra tecla de cambio.

1.1. Teclas del cursor

Las teclas del cursor se diferencian de las demás teclas porque su comportamiento depende de que aparezca actualmente en pantalla o no un cursor. A continuación resumimos el comportamiento cuando aparece un cursor en pantalla.

TECLA	SIN TECLA DE CAMBIO	CON TECLA DE CAMBIO DERECHA
◀	Desplaza el cursor a la izquierda	Desplaza el cursor al principio
▶	Desplaza el cursor a la derecha	Desplaza el cursor al final
▼	Desplaza el cursor hacia abajo	Desplaza el cursor a la parte inferior (final)
▲	Desplaza el cursor hacia arriba	Desplaza el cursor a la parte superior (principio)

Cuando no aparezca el cursor en pantalla, al pulsar cualquiera de estas teclas, se ejecutará otra operación indicada.

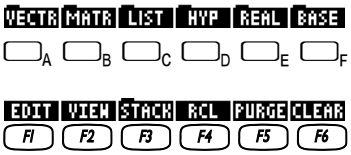
◀ Muestra en pantalla el dibujo editado anteriormente.
▶ Intercambia los objetos de los niveles 1 y 2 de la pila

↑ Entra en la aplicación pila interactiva
↓ o loca el objeto del nivel 1 en su mejor modo de visualización
← Elimina el objeto del nivel 1 de la pila
DEL Elimina todos los objetos de la pila

Cuando está encendida la calculadora ON se convierte en la tecla cancel (cancela la actividad actual que se realiza). Por ejemplo para cancelar un programa que se está ejecutando, pulsamos Cancel.

2. Manejo de menús

Los menús de teclas (SOFT menú) asocian las etiquetas en la parte inferior de la pantalla con las seis teclas en la primera fila del teclado (F1 a F6). Presionando la tecla apropiada del menú, la función en la etiqueta asociada se activa o se ejecuta.

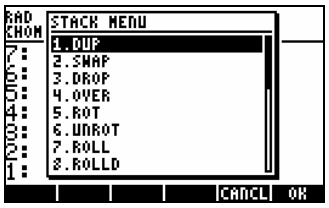


Las seis etiquetas asociados con las teclas F1 a F6 forman parte de un menú de funciones de la calculadora. Dado que la calculadora solamente tiene seis teclas del menú, solo se muestran seis etiquetas a la vez. Sin embargo el menú puede tener más de seis opciones. Cada grupo de 6 opciones se conoce como una página de menú. Para mostrar la siguiente pagina de menú (si existe), presionamos la tecla NXT Para pasar a la pagina anterior (menú anterior), pulsamos ← NXT

Paso 1: ← PRG *(Ir al menú de programación)*
Paso 2: NXT NXT *(Ir al final del menú de programación)*

Paso 3: ← PREV *(Ir a la pagina 2 del menú de programación)*

Sin embargo, los menús de teclas no son la única manera de acceder a las funciones en la calculadora. La manera alternativa será referida como menús de listas (CHOOSE boxes).



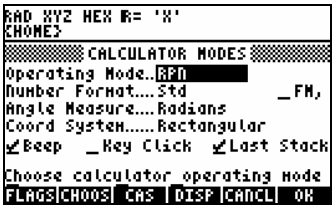
En esta ventana se muestra el mismo menú de programación como menú de listas. Es muy engorroso trabajar con este tipo de menús, muy recomendable trabajar con el soft menú es más sencillo y practico.

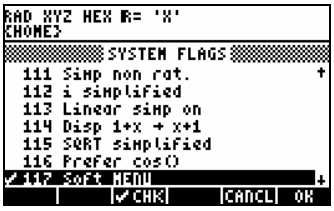
Nosotros podemos seleccionar el formato en el cual lo menús serán mostrados cambiando las banderas o señales del sistema de la calculadora (la bandera o señal del sistema es una variable de la calculadora que controla cierta operación o modo de la calculadora). La bandera 117 del sistema se puede fijar para producir ya sea un menú de teclas (SOFT menú) o un menú de listas (CHOOSE boxes). Para tener acceso a esta bandera seguimos los siguientes pasos:

Pasos a seguir para elegir SOFT MENU

Paso 1: MODE ▲ ↰ ▲ ▼

Paso 2:





La gran mayoría de los ejemplos de este libro se demuestran usando SOFT MENU.

También podemos ir al menú PRG, colocando en la pila 22 MENU ENTER

3. Modos Algebraico y RPN

La calculadora se puede operar en dos modos diferentes, el modo de notación polaca reversa (RPN) y el modo algebraico (ALG).

3.1. Modo Algebraico

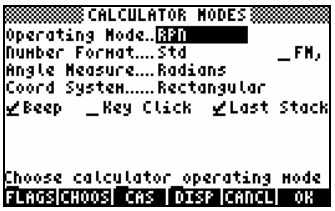
Este modo se asemeja a la manera en que uno escribe expresiones aritméticas en el papel. Los cálculos se realizan introduciendo los argumentos después del comando, es decir que en la mayoría de las operaciones significa introducir números, funciones y operadores en el mismo orden que escribimos. A continuación exponemos un ejemplo.

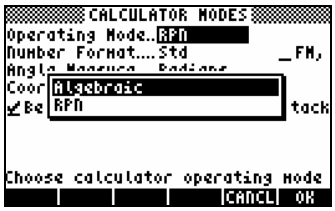
Ejemplo 01: *Realizar la suma de dos números enteros en modo ALG*

Paso 1: [2] [+] [3] *(Ingresamos los números)*
Paso 2: [ENTER] *(Enter nos mostrara el resultado)*

En la HP49 el modo algebraico es el modo por defecto, esto solo para usuarios principiantes ya que un usuario con experiencia podría preferir el modo RPN.

Pasos a seguir para fijar el modo ALG
Paso 1: [MODE]
Paso 2: [▼]





3.2. Modo RPN

La notación polaca inversa, también llamada notación postfija, es un método de introducción de ordenes alternativo a la notación algebraica también se usa en algunos lenguajes como PostScript o Forth.

En 1920 Jan Lukasiewicz ideó un método para escribir expresiones matemáticas sin utilizar ni paréntesis ni corchetes llamada notación polaca. En 1972 HP se basó en él e incorporo en su primera calculadora científica de bolsillo, la HP35, la notación polaca inversa.

Este sistema no es el habitual de las calculadoras pero tiene varias ventajas:

- Ahorra pulsaciones de teclas, con lo que se introducen los datos más rápido.
- Permite ver los resultados intermedios, con lo que se perciben más fácilmente los errores.
- Se parece a como se calcula con papel y lápiz.

El RPN (por sus siglas en inglés) se basa en el concepto de pila de datos. Primero se introducen los datos en la pila y luego se indica la orden a realizar. Se puede ver mejor con ejemplos.

Ejemplo 02: *Realizar la siguiente operación 85 -31*

Paso 1: [8] [5] [SPC] [3] [1] [−]

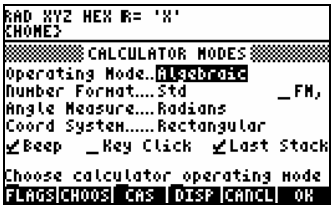
Ejemplo 03: *Realizar la siguiente operación* $23^2-(13\times 9)+\frac{5}{7}$

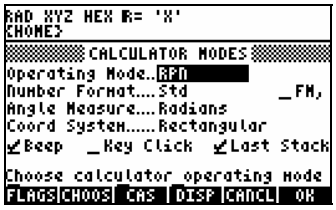
Paso 1: [2] [3] [↰] [x^2]
Paso 2: [1] [3] [SPC] [9] [×]
Paso 3: [−] [5] [SPC] [7] [÷]
Paso 3: [+]

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

Pasos a seguir para fijar el modo RPN

Paso 1: MODE
Paso 2: +/- ▦





4. Manejo de la pila

Una pila, es una estructura de datos que consta de una serie de objetos (datos), el cual las inserciones y eliminaciones se hacen por su extremo. Es una estructura LIFO (Last In - First Out) Ultimo que llega primero en salir. Por eso en nuestra calculadora se denomina Pila a la entrada y salida de objetos.

Los conceptos fundamentales de las operaciones de la pila son:

- Un comando que necesita argumentos (objetos sobre los que actúa el comando) y que toma sus argumentos de la pila. Por tanto estos deberán estar presentes antes de ejecutar el comando.
- Los argumentos de un comando se borran de la pila cuando se ejecuta el comando.
- Los resultados se devuelven a la pila para que puedan verse y utilizarse de nuevo en otras operaciones.

La pila es una serie de ubicaciones de almacenamiento en la memoria para números y otros objetos. Dichas ubicaciones se llaman nivel 1, 2, 3, etc. El número de niveles varía de acuerdo con la cantidad de objetos almacenados en la pila.

4.1. Menú de comandos de la pila

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

COMANDO	DESCRIPCION
CLEAR	Borra la Pila
DUP	Duplica el objeto del nivel 1
SWAP	Invierte dos objetos de la pila del nivel 1 y 2
DROP	Borra el primer objeto de la pila
OVER	Devuelve una copia del objeto del nivel 2
ROT	Hace girar los 3 primeros objeto = 3 ROLL
UNROT*	Desplaza el objeto del nivel 1 al nivel 3 de la pila
ROLL	Desplaza el objeto del nivel n+1(n esta en el nivel 1)
ROLLD	Desplaza hacia abajo una parte de la pila entre el nivel 2 y el nivel n+1 (n esta en el nivel 1)
PICK	Copia el objeto del nivel n+1 al nivel 1 (n esta en el nivel 1)
PICK3*	Copia el objeto del nivel 3 al nivel 1 de la pila
DEPTH	Devuelve el número del objetos de la pila
DUP2	Duplica 2 objetos del nivel 1y 2
DUPN	Duplica n objetos en la pila comenzando por el nivel 2 (n esta en el nivel 1)
DROP2	Borra los objetos de los niveles 1 y 2
DROPN	Borra los primeros objetos n+1 de la pila (n está en el nivel 1)
DUPDUP*	Duplica un objeto del nivel 1 dos veces
NIP*	Borra el objeto del nivel 2 de la pila
NDUPN*	Duplica n veces el objeto del nivel 2 (n debe estar en el nivel 1)

* Comandos disponibles solo en la HP49

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

(STACK)


TOOL

También podemos ingresar al menú de comandos STACK, escribiendo en la pila: `73 MENU` ENTER

Ejemplo 04: *Duplicar un objeto ubicado en el nivel 1 de de la pila*

Paso 1: `13` ENTER
Paso 2: ◀ ▲ NXT DUP
(DUP también equivale al ENTER)

Paso 1: `13` ENTER
Paso 2: TOOL STACK DUP

Ejemplo 05: *Cambia los objetos de los niveles 1 y 2 utilizando SWAP*

Paso 1: `1906` SPC `2006` ENTER
Paso 2: TOOL STACK SWAP *(SWAP equivale a la tecla* ▶*)*

Ejemplo 06: *Coloca un objeto del nivel 4 al nivel 1 de la pila (Uso de ROLL, n tiene que estar en el nivel 1), (n+1)=4: entonces n=3.*

Paso 1: `3` ENTER
Paso 2: ◀ ▲ ROLL

Paso 1: `3` ENTER
Paso 2: TOOL STACK ROLL

4.2. La pila interactiva

Es un entorno especial en el que se vuelve a definir el teclado para un conjunto específico de operaciones de manipulación de la pila, esta permite hacer lo siguiente:

- Copiar y desplazar objetos a niveles diferentes.
- Borrar objetos de la pila.
- Editar objetos de la pila.
- Desplazar la ventana para ver el resto de la pila.
- Copiar el contenido de cualquier nivel de la pila a línea de comandos

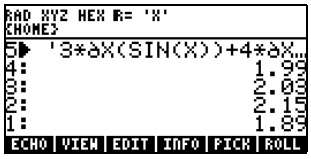
Cuando se activa la pila interactiva, el puntero ▶ de la pila también se activa (señalando al nivel actual de la pila)

COMANDO	DESCRIPCION
ECHO	Copia el contenido del nivel actual a la posición del cursor de la línea de comandos.
VIEW	Visualiza o edita el objeto del nivel actual utilizando el entorno más adecuado.
EDIT	Edita el nivel actual con el editor más apropiado
INFO	Muestra información sobre el objeto del nivel actual, incluye su tamaño en bytes.
PICK	Copia el contenido del nivel actual al nivel 1
ROLL	Mueve el contenido del nivel actual al nivel 1 y desplaza hacia arriba la parte de la pila que se encuentra por debajo del nivel actual.
ROLLD	Mueve el contenido del nivel 1 al nivel actual y desplaza hacia abajo la parte de la pila que se encuentra por debajo del nivel actual.
→LIST	Crea una lista que contiene todos los objetos desde el nivel 1 hasta el nivel actual.
DUPN	Duplica los niveles comprendidos entre el nivel 1 y el nivel actual. (Equivale a DROPN)
DRPN	Borra todos los niveles desde el nivel 1 hasta el nivel actual.
KEEP	Borra todos los niveles superiores al nivel actual.
GOTO	Selecciona un nivel de la pila, para el nivel actual.
LEVEL	Introduce el número del nivel actual en el nivel 1.

Ejemplo 07: *Utiliza la pila interactiva para copiar un objeto del ultimo nivel al nivel 1 de la pila*

Paso 1. ▲ ▲ ▲ ▲ ▲
Paso 2: [1][2][3] ENTER

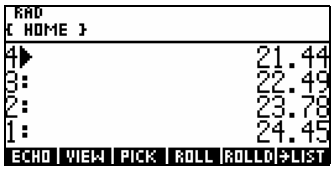
Paso 1: ▲ ▲ ▲ ▲ ▲
Paso 2: [1][2][3] ENTER



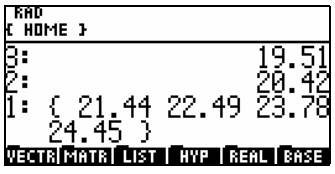
Ejemplo 08: *Utiliza la pila interactiva para crear una lista con los cuatro últimos objetos ingresados en la pila.*

Paso 1.Los objetos están en la pila:

Paso 2: ▲ ▲ ▲ ▲



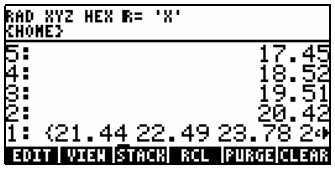
Paso 3: [1][2][3] ENTER



Paso 1.Los objetos están en pila:

Paso 2: ▲ ▲ ▲ ▲

Paso 3: NXT [1][2][3] ENTER



5. Tipo de objetos

Los elementos básicos de información utilizados se denominan objetos. Por ejemplo, un número real, una ecuación y un programa son cada uno de ellos un objeto. Un objeto ocupa un solo nivel de la pila y puede almacenarse en una variable. La siguiente tabla describe algunos de estos.

TIPO	OBJETO	EJEMPLO
0	Número real	81.85
1	Número complejo	(3,4)
2	Cadena de caracteres “String”	"INGENIERIA CIVIL"
3	Vector Real	[4 1 -3]
3	Forma polar de un Vector	[5 ∠30]
3	Matriz real	[[3 5 -1] [4 -8 3]]
4	Vector Complejo	[(1,2) (2,5)]
5	Lista	{ "FNI" 1 9 0 6 }
6	Nombre de Variable	'A'
7	Variable en uso	'i' « FOR i 1 + NEXT »
8	Programa	« 2 ^ »
9	Constante simbólica	'π'
9	Expresión algebraica	'X^2+2*X+3'
10	Número binario	#240d
11	Objeto gráfico (Grob)	Graphic 131 × 64
12	Objeto etiquetado	Area:126
13	Objeto de Unidad	110_km
14	Nombre de XLIB	XLIB 755 1
15	Directorio	DIR A B 2 ...END
16	Librería	Library 1659: MetNum
17	Objeto de Reserva	Backup MYDIR
18	Funciones incorporadas	SIN, COS
19	Comandos de la HP	IF, FOR, START, DRAW
25	Objeto codificado	Code
26	Datos de librería	Library Data
30	Objeto externo	External

Una forma de saber el número de tipo, del objeto que se encuentra en la pila es utilizando el comando TYPE

Ejemplo 9: *Determina el tipo de objeto del nivel 1 de la pila*

El comando TYPE devuelve el número de tipo de objeto que se encuentra en una variable, para esto se necesita el nombre de la variable.

Como es muy importante saber con que tipo de datos trabajamos para programar a continuación describimos los comandos del menú TYPE

COMANDO	DESCRIPCION
OBJ→	Descompone un objeto compuesto (*x*) en sus componentes.
→ARRY	Combina los números en un sistema (vector o matriz)
→LIST	Combina los objetos entre el nivel 1 y el nivel actual en una lista. Necesita el número de objetos.
→STR	Convierte un objeto (*x*) en una cadena.
→TAG	Define (etiqueta) un objeto (*y*) mediante un nombre o una secuencia descriptiva (*x*)
→UNIT	Crea un objeto de unidades de medida a partir de un número real (*y*) y la parte de la unidad de un objeto de unidades de medida (*x*).
C→R	Descompone un número complejo (*x*) en dos números reales.
R→C	Combina los componentes separados real (*y*) e imaginario (*x*) para formar un número complejo.
NUM	Devuelve el número del código de un carácter.
CHR	Convierte un carácter (símbolo) a una cadena (número)
DTAG	Elimina todas las etiquetas de identificación de un objeto (*x*).
EQ→	Descompone una ecuación a partir del signo de igualdad.

6. Almacenar, Recuperar, Borrar objetos (variables)

Las variables u objetos en la calculadora son similares a los archivos en el disco duro de un ordenador (computadora). Es posible almacenar un objeto (valores numéricos, expresiones algebraicas, listas, vectores, matrices, programas, etc.) en una variable. Las variables se identifican por un nombre (pueden contener hasta 127 caracteres), el cual puede ser cualquier combinación de caracteres alfabéticos o numéricos, comenzando siempre por una letra.

No se puede asignar a una variable un nombre igual al de una función en la calculadora. Los nombres reservados por la calculadora son los siguientes: ALRMDAT, CST, EQ, EXPR, IERR, IOPAR, MAXR, MINR, PICT, PPAR, PRTPAR, VPAR, ZPAR, der_, e, i, n1,n2,..., s1, s2,..., ΣDAT, ΣPAR, π, ∞.

Ejemplo 10: *Algunos nombres válidos para una variable*

```
'A'        'A1'        'α'        'ΔT'
'a'        'AB"'       '■β'       '←LIB→'
'z1'       'Nmin'      'χ²2'      '↑DIR'
'■F'       'AREA'      'ωS'       '→BORRAR'
```

6.1. Almacenar un objeto

Después de entrar un objeto en la pila, este se lo puede almacenar (guardar) en una variable asignándole un nombre. Todas las variables que creamos se almacenan en el menú del usuario [VAR]
Para almacenar un objeto se sigue la siguiente sintaxis:

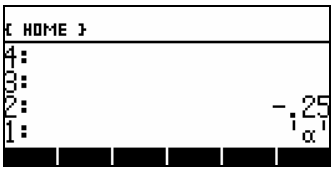
```
2:  Objeto
1:  'Nombre'
     STO▶
```

```
2:                    (1 2 3)
1:                  'LISTA'
EDIT|VIEW|STACK| RCL |PURGE|CLEAR
```

Ejemplos 11: *Almacenar* `-0.25` *con el nombre* `'α'`

Paso 1. Escribir el objeto: [0] [.] [2] [5] [+/-] [ENTER]
Paso2. Escribir el nombre: [α] [→] [A] [ENTER]
Paso3. Almacenar el objeto: [STO] *(Almacena el objeto -0.25 en la variable α)*
Paso4. Ver el objeto: [VAR] *(Menú del usuario)*

Paso 1. Escribir el objeto: [0] [.] [2] [5] [+/-] [ENTER]
Paso2. Escribir el nombre: [ALPHA] [→] [FI] [ENTER]
Paso3. Almacenar el objeto: [STO▶]
Paso3. Ver el objeto: [VAR]

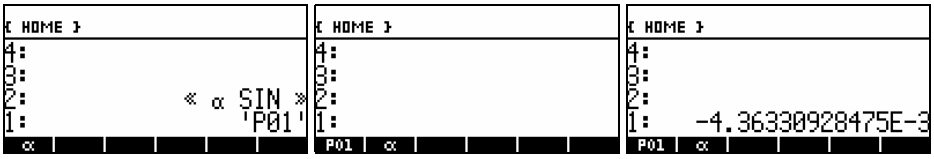


La constante almacenada podrá ser utilizada en cualquier operación, únicamente pulsando la tecla correspondiente al menú de pantalla [VAR] [α]

Ejemplo 12: *Almacenar un programa que halle el seno de la variable α (ejemplo anterior), con el nombre de* `'P01'`

Paso 1. Escribir el programa: [←] [−] [▭] [SIN] [ENTER]
Paso 2. Escribir el nombre: [α] [◀] [0] [1] [ENTER]
Paso 3. Almacenar el objeto: [STO]
Paso 4. Ejecutar el programa: [VAR] [P01]

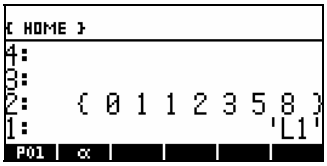
Paso 1. Escribir el programa: [→] [+] [FI] [SIN] [ENTER]
Paso 2. Escribir el nombre: [ALPHA] [P] [0] [1] [ENTER]
Paso 3. Almacenar el objeto: [STO▶]
Paso 4. Ejecutar el programa: [VAR] [P01]



Ejemplo 13: *Almacenar la lista* `{ 0 1 1 2 3 5 }` *con el nombre* `'L1'`

Paso 1. Escribir el objeto:
[←] [+] [0] [SPC] [1] [SPC] [1] [SPC] [2] [SPC] [3] [SPC] [5] [ENTER]
Paso 2. Escribir el nombre: [α] [NXT] [1] [ENTER]
Paso 3. Almacenar el objeto: [STO]

Paso 1. Escribir el objeto:
[←] [+] [0] [SPC] [1] [SPC] [1] [SPC] [2] [SPC] [3] [SPC] [5] [ENTER]
Paso 2. Escribir el nombre: [ALPHA] [L] [1] [ENTER]
Paso 3. Almacenar el objeto: [STO▶]

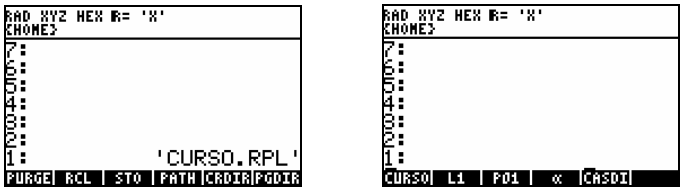


Para crear un directorio hacemos uso del comando CRDIR

Ejemplo 14: *Crear un directorio con el nombre* `'CURSO.RPL'`

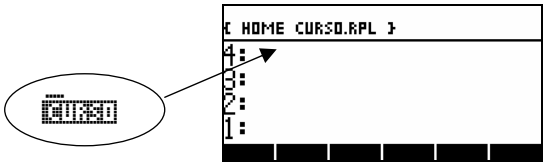
Paso 1. Escribir el nombre: [α] [⬭] [TAN] [▶] [SIN] [EVAL] [·] [▶] [◀] [NXT] [ENTER]
Paso 2. Almacenar el objeto: [←] [VAR] [DIR] [CRDIR]

Paso 1. Escribir el nombre: [ALPHA] [C] [U] [R] [S] [O] [·] [R] [P] [L] [ENTER]
Paso 2. Almacenar el objeto: [←] [PRG] [TIME] [DIR] [CRDIR]



Otra forma de hacer uso del comando CRDIR es escribiendo en la pila el comando CRDIR ENTER

Para ingresar al directorio solo vamos a VAR y pulsamos CURSO



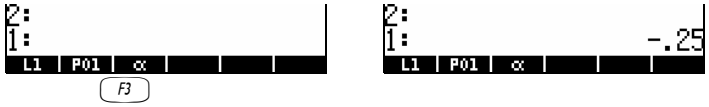
Para borrar un directorio hacemos uso del comando PURGE

6.2. Recuperar un objeto

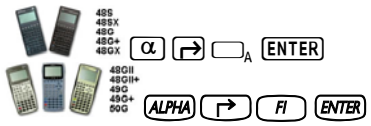
Para recuperar un objeto del menú VAR (el menú VAR, contiene todos los objetos definidos por el usuario), consiste en presionar la tecla correspondiente al nombre del objeto del menú VAR

Ejemplo 15: *Recuperar el objeto almacenado en el ejemplo 12*

Buscamos en el menú VAR el nombre del objeto α si no se encuentra en la Pág.1 extendemos a la pág.2 con NXT (si existiera) hasta encontrar el nombre del objeto α una vez encontrado, pulse el mismo con la tecla correspondiente al nombre del objeto α



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



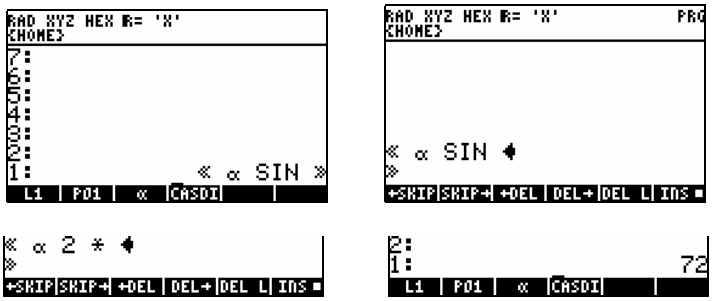
6.3. Guardar un objeto editado

La forma más fácil de colocar el objeto en la pila para su edición, es, solo presionando ↱ α (tecla correspondiente al menú de la variable). Y una vez editado lo guardamos con el siguiente atajo ↰ α

Ejemplo 16: *Editar el programa del ejemplo 13, que multiplique por dos la variable* α *,*

Paso 1. Colocamos el programa en la pila: ↱ 20
Paso 2. Editamos el programa: ▼ ↱ ▶ EDIT 2 × ENTER
Paso 3. Guardamos a la misma variable: ↰ 20
Paso 4. Ejecutar el programa: VAR 20

Paso 1. Colocamos el programa en la pila: ↱ 20
Paso 2. Editamos el programa: ▼ ↱ ▶ EDIT 2 × ENTER
Paso 3. Guardamos a la misma variable: ↰ 20
Paso 4. Ejecutamos el programa: VAR 20



Ejemplo 17: *Editar la variable* 'α' *asignarle (guardar) otro valor.*

Paso 1: Escribir otro valor: 3 6 ENTER
Paso 2. Guardar: ↰ α

Como para los ejemplos anteriores hacemos un pequeño cuadro para editar asignar valores a las variables:

FUNCIÓN	DESCRIPCIÓN
→ ▒P01▒ ▼ ← ▒P01▒	Coloca el valor de P01 en la pila Edita con el mejor visor posible Guarda un nuevo valor (si se edita) en P01

P01 puede ser cualquier variable.

6.4. Borrar un objeto

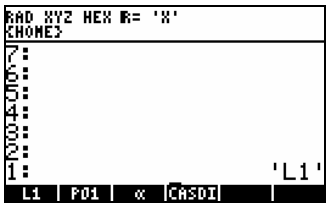
Para borrar un objeto del menú VAR en el directorio actual, primero se ingresa el nombre del objeto en la pila, luego, pulse DROP

```
1: 'NombreObjeto'
DROP
```

Ejemplo 18: *Borrar el objeto L1*

Paso 1. Escribir el nombre del objeto: ' α NXT 1 ENTER
Paso 2. Borrar el objeto: ← EEX

Paso 1. Escribir el nombre del objeto: → ALPHA NXT / ENTER
Paso 2. Borrar el objeto: TOOL DROP



Para borrar más de un objeto, se coloca en una lista los objetos a borrar, seguimos la siguiente sintaxis:

```
1: { Objeto1 Objeto2 ... Objeto N }
```

DROP

Ejemplo 19: *Borrar* `P01` *y* `α` *del menú VAR*

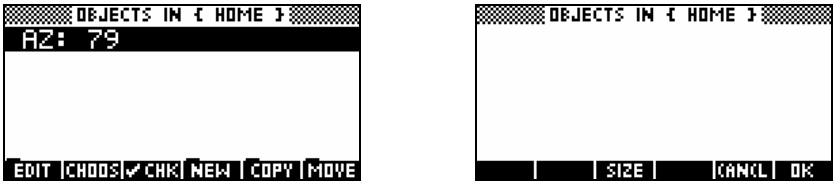
Paso 1. Colocamos los nombres de los objetos en una lista:
◁ + 2 0 ▸ ENTER
Paso 2. Borrar la lista de objetos: ◁ EEX

Paso 1. Colocamos los nombres de los objetos en una lista:
◁ + 2 0 ▸ ENTER
Paso 2. Borrar la lista de objetos: TOOL PURGE

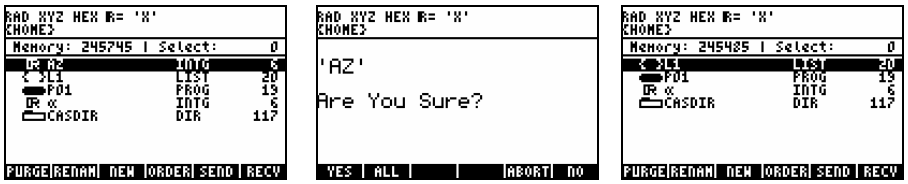
Otra forma de borrar y crear variables e incluso directorios es utilizando el menú FILES (Administrador de objetos).

Ejemplo 20: *Borrar el objeto* `AZ`

Paso 1. Ingresar al localizador de variables: ▷ MEMORY
Paso 2. Seleccionar la variable que se desea borrar
Paso 3. Borramos la variable: NXT PURGE OK



Paso 1. Ingresamos al menú FILES: ◁ APPS 1:
Paso 2. Seleccionamos la variable a borrar
Paso 3. Aceptamos la eliminación de la variable: NXT PURGE YES



7. Funciones de comparación o test

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

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

Como se puede apreciar SAME y == cumplen la misma función.
Para tener acceso a estos comandos del menú Test:

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

Ejemplo 21: *Compare si dos números puestos en la pila son iguales.*

Paso 1: 5 SPC 3 ENTER
Paso 2 (Test): PRG TEST ==

Se ha utilizado la función == comparando si los objetos puestos en la pila son iguales, el resultado nos da `0` (FALSO).

8. Funciones lógicas

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

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

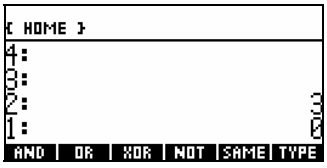
Condición 1	Condición 2	AND	OR	XOR
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

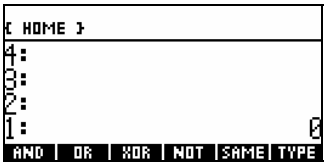
NOT: Siempre devuelve el inverso lógico

Condición	NOT
1	1
0	1

Ejemplo 22: *si las variables a, b, c, d contienen los valores 120, 6, 3, 0 respectivamente, cual es el valor de la sgte. expresión:* C D AND A B == OR

Paso 1: 3 SPC 0 ENTER





Paso 2: [1] [2] [0] [SPC] [6] [ENTER] [▒▒]

Paso 2: [▒▒]

9. Indicadores de sistema

Los indicadores del sistema (Banderas: `FLAGS`) nos permite activar funciones de configuración de la calculadora, estos nos son muy útiles en programación:

COMANDO	DESCRIPCION
`SF`	Activa el indicador especificado del sistema
`CF`	Desactiva el indicador especificado del sistema
`FS?`	Da verdadero (1) si el indicador esta activado y falso (0) si el indicador esta desactivado
`FC?`	Da verdadero (1) si el indicador esta desactivado y falso (0) si el indicador esta activado
`FS?C`	Prueba el indicador especificado, da verdadero (1) si esta el indicador esta activado y falso (0) si el indicador esta desactivado y luego lo desactiva
`FC?C`	Prueba el indicador especificado, da verdadero (1) si esta el indicador esta desactivado y falso (0) si el indicador esta activado y luego lo desactiva

Ejemplo 23: *Se quiere verificar si los argumentos simbólicos devuelven números, si no lo es, activarlo. Para este ejemplo el indicador es el número* `-3`
Verificamos con `FS?`

Paso 1: [3] [+/-] [ENTER]
Paso 2: [←] [CST] [▒▒▒▒] [▒▒]

Paso 1: [3] [+/-] [ENTER]
Paso 2: [←] [PRG] [⊥] [▒▒▒] [NXT] [NXT] [▒▒]

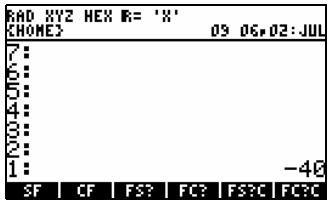
El resultado no da `0` (Falso) entonces lo activamos con `SF`:

Paso 3: 3 +/- ENTER
Paso 4: ↶ CST ▒▒▒▒ ▒▒▒

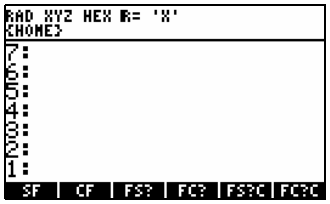
Paso 3: 3 +/- ENTER
Paso 4: ↶ PRG ▒▒▒▒ NXT NXT ▒▒▒

Ejemplo 24: *Se desea desactivar la visualización del reloj en la parte superior de la pantalla*

Paso 1: 4 0 +/- ENTER
Paso 2: ↶ CST ▒▒▒▒ ▒▒▒



Paso 1: 4 0 +/- ENTER
Paso 2: ↶ PRG ▒▒▒▒ NXT NXT ▒▒▒



Ejemplo 25: *Activar los indicadores -60 -117 (HP49)*
-60 Este indicador activa `[α] locks Alpha` (HP48/49)
-117 Este indicador activa `Soft MENU` (HP49)

Paso 1: ↶ + 6 0 +/- ENTER
Paso 2: ↶ CST ▒▒▒▒ ▒▒▒

Paso 1: ↶ + 6 0 +/- SPC 1 1 7 +/- ENTER
Paso 2: ↶ PRG ▒▒▒▒ NXT NXT ▒▒▒

Para reconfigurar todos los indicadores con sus valores por defecto hacemos uso del comando RESET

↶ NXT ▒▒▒▒ CST ▒▒▒▒

MODE NXT ▒▒▒▒

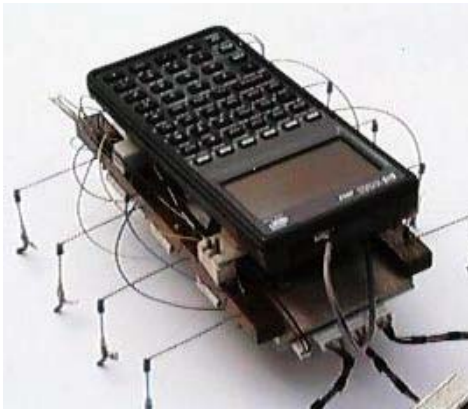
Capítulo II

Resolución de problemas con la calculadora

El ingeniero de este siglo trabaja en un entorno que requiere muchas habilidades y capacidades no técnicas. Aunque la computadora es la herramienta del cálculo primaria de la mayoría de los ingenieros, también es para adquirir capacidades no técnicas adicionales.
Los ingenieros requieren firmes habilidades de comunicación tanto para presentaciones orales como para preparar materiales escritos. Las computadoras proporcionan software que ayuda a escribir sinopsis y elaborar materiales y gráficas para presentaciones e informes técnicos. El correo electrónico *(e-mail)* y la Word Wide Web *(WWW)* también son importantes canales de comunicación, también donde podemos encontrar los mejores recursos para solucionar un problema.

Un ingeniero resuelve problemas, pero los problemas no siempre se formulan con cuidado. El ingeniero debe ser capaz de extraer un enunciado de problema de un análisis del mismo y luego determinar las cuestiones importantes relacionadas con él. Esto implica no sólo crear un orden, sino también aprender a correlacionar el caos; no sólo significa analizar los datos, sino también sintetizar una solución. La integración de ideas puede ser tan importante como la descomposición del problema en fragmentos manejables. La solución a un problema podría implicar no sólo un razonamiento abstracto sobre el problema, sino también aprendizaje experimental a partir del entorno del problema.

Desde la invención de la computadora a fines de la década de 1950, han ocurrido varios avances muy significativos en ingeniería. La invención del microprocesador, una diminuta computadora más pequeña que un sello de correo, es uno de los logros culminantes en ingeniería. Los microprocesadores se emplean en equipos electrónicos, aparatos domésticos, juguetes y juegos, así como en automóviles, aviones y transbordadores espaciales, porque ofrecen capacidades de cómputo potentes pero económicas. Además, los microprocesadores proporcionan la potencia de cómputo a las calculadoras y computadoras personales.



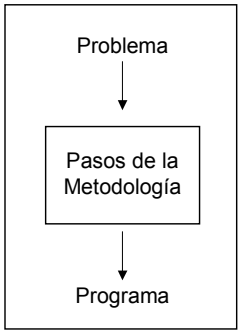
La invención del microchip permitió reducir el tamaño de los ordenadores, primero lo suficiente para colocarlos encima de la mesa, y más tarde para llevarlos en la mano. Los dispositivos de mano más completos disponen de varios megabytes (millones de caracteres) de espacio para almacenar archivos, enorme capacidad de cálculo, con utilidades de hoja de cálculo y gráficos, y los medios necesarios para enviar y recibir correo electrónico y recorrer Internet.
En la fotografía se muestra un ROBOT NITI, Robot araña de seis patas con tres sensores infrarrojo NITINOL PIC16F84
Protocolo RS232 HP 48 GX

10. Metodología para resolver problemas

La resolución de problemas es una parte clave de los cursos de ingeniería, y también de los de ciencias de la computación (MAT1104, SIS1100, SIS1101), matemáticas, física y química. Por tanto, es importante tener una estrategia consistente para resolver los problemas. También es conveniente que la estrategia sea lo bastante general como para funcionar en todas estas áreas distintas, para no tener que aprender una técnica para los problemas de matemáticas, una técnica diferente para los problemas de física, etc. La técnica de resolución de problemas que se presenta funciona para problemas de ingeniería y puede adaptarse para resolver también problemas de otras áreas; sin embargo, da por hecho que vamos a usar una calculadora para ayudarnos a resolver el problema.

El desarrollo de un programa que resuelva un problema dado es una tarea compleja, ya que es necesario tener en cuenta de manera simultánea muchos elementos. Por lo tanto, es indispensable usar una metodología de programación.

Una metodología de programación es un conjunto o sistema de métodos, principios y reglas que permiten enfrentar de manera sistemática el desarrollo de un programa que resuelve un problema algorítmico. Estas metodologías generalmente se estructuran como una secuencia de pasos que parten de la definición del problema y culminan con un programa que lo resuelve.



La principal razón para que las personas aprendan lenguajes de programación es utilizar una computadora o calculadora como una herramienta para la resolución de problemas. Dos fases pueden ser identificadas en el proceso de resolución de problemas.

1.- Fase de resolución del problema.
2.- Fase de implementación (realización del programa) en la calculadora.

El resultado de la primera fase es el diseño de un algoritmo para resolver el problema. La ejecución y verificación del programa en una computadora es el objetivo final de la fase de implementación o realización.

No existe un método universal que permita resolver cualquier problema. En general, la resolución de problemas es un proceso creativo donde el conocimiento, la habilidad y la experiencia tienen un papel importante. El proceder de manera sistemática (sobre todo si se trata de problemas complejos) puede ayudar en la solución.

Las fases mencionadas anteriormente disponen de una serie de pasos que enlazados convenientemente conducirán a la solución del problema. Aunque el diseño de programas es un proceso esencialmente creativo, se pueden considerar una serie de fases o pasos comunes que generalmente deben seguir todos los programadores.

Las fases de resolución de un problema con computadora o calculadora son:

1. Definición del problema
2. Análisis del problema
3. Diseño del algoritmo

1

4. Codificación
5. Prueba y depuración
6. Documentación
7. Mantenimiento

2

Antes de conocer las tareas a realizar en cada fase, definiremos el concepto y significado de algunos conceptos básicos:

10.1. Algoritmo

Un algoritmo es un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema. De un modo más formal, un algoritmo es una secuencia finita de operaciones realizables, no ambiguas, cuya ejecución da una solución de un problema en un tiempo finito.

Las características más relevantes de los algoritmos son:

- Finito: Un algoritmo debe ser siempre terminar después de un número finito de pasos.

- Definido: Cada Paso de un algoritmo debe ser definido en forma precisa, estableciendo las acciones que van a efectuar clara y rigurosamente en cada caso.
- Entradas: El algoritmo tiene cero o más entradas, es decir cantidades qeu se entregan inicialmente al algoritmo antes de su ejecución.
- Salidas: Un algoritmo tiene una o más salidas, es decir cantidades que tiene una relación específica respecto a las entradas.
- Efectivo: Generalmente, también se espera que un algoritmo sea efectivo. Esto significa que todas las operaciones ha ser realizadas en el algoritmo deben ser lo suficientemente básicas de modo que se puedan en principio ser llevadas a cabo en forma exacta y en un periodo de tiempo finito por una persona usando lápiz y papel.

En la práctica, para evaluar un buen algoritmo se considera el tiempo que requiere su ejecución, esto puede ser expresado en términos del número de veces que se ejecuta cada paso. Otros criterios de evaluación pueden ser la adaptabilidad del algoritmo al computador, su simplicidad y elegancia, etc. Algunas veces se tienen varios algoritmos para solucionar el mismo problema, y se debe decidir cual es el mejor. Esto último conduce al "Análisis de algoritmo". Dado un algoritmo es determinar sus características de desempeño. El análisis y el diseño del algoritmo requieren la descripción del problema a base de refinamientos sucesivos conocidos como lenguajes algorítmicos.

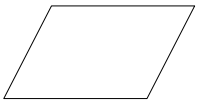
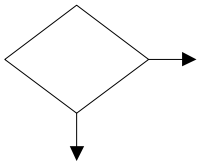
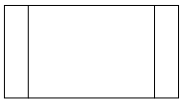
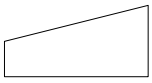
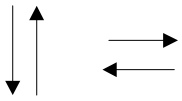
Los lenguajes algorítmicos son una serie de símbolos y reglas que se utiliza para describir de manera explicita un proceso, estos lenguajes son:

Gráficos: Es la representación gráfica de las operaciones que se realiza un algoritmo (Diagramas de flujo).

No gráficos: Representa en forma descriptiva las operaciones que debe realizar un algoritmo (Pseudocódigo).

10.2. Diagrama de flujo

Un diagrama de flujo es la representación gráfica de un algoritmo. También se puede decir que es la representación detallada en forma gráfica de como deben realizarse los pasos en la computadora o calculadora para producir resultados. Esta representación gráfica se da cuando varios símbolos (que indican diferentes procesos), se relacionan entre si mediante líneas que indican el orden en que se deben ejecutar los procesos.

SIMBOLO	DESCRIPCION
	Indica el inicio y el final de nuestro diagrama de flujo
	Indica la entrada y salida de datos (E/S)
	Símbolo de proceso y nos indica la asignación de un valor en la memoria y/o la ejecución de una operación
	Símbolo de decisión, indica la realización de una comparación de valores
	Símbolo utilizado para representar subrutinas o un proceso predeterminado
	Conector dentro de página, representa la continuidad del diagrama dentro la misma página
	Conector fuera de página, representa la continuidad del diagrama en otra página
	Indica la salida de información por impresora (se utilizar en ocasiones en lugar del símbolo de E/S)
	Entrada manual (teclado) (Se utilizar en ocasiones en lugar del símbolo de E/S)
	Líneas de flujo o dirección, indican la secuencia en que se realizan las operaciones.

Recomendaciones para el diseño de flujogramas:

- Todo texto escrito dentro de un símbolo deberá ser escrito claramente, evitando el uso de muchas palabras.
- Se deben de usar solamente líneas de flujo horizontales y/o verticales.
- Se debe evitar el cruce de líneas utilizando conectores.
- Se debe usar conectores solo cuando sea necesario.
- No deben quedar líneas de flujo sin conectar.
- Se deben trazar los símbolos de manera que se pueda leer de arriba hacia abajo y de izquierda a derecha.

10.3. Pseudocódigo

Mezcla de lenguaje de programación y español (o inglés o cualquier otro idioma) que se emplea, dentro de la programación estructurada, para realizar el diseño de un programa. En esencial, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos. Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El pseudocódigo utiliza palabras que indican el proceso a realizar.

Ventajas de utilizar un pseudocódigo a un diagrama de flujo:

- Ocupa menos espacio en una hoja de papel.
- Permite representar en forma fácil operaciones repetitivas complejas.
- Es muy fácil pasar el pseudocódigo a un programa en algún lenguaje de programación.
- Si se sigue las reglas se puede observar claramente los niveles que tiene cada operación.

11. Resolución de problemas utilizando la calculadora

11.1. Definición del problema

Esta fase esta dada por el enunciado del problema, el cual requiere una definición clara y precisa. Es importante que se conozca lo que se desea que realice la calculadora; mientras esto no se conozca o entienda del todo no tiene mucho caso continuar con la siguiente etapa.

11.2. Análisis del problema

Una vez que se ha comprendido lo que debe hacer nuestro programa en la calculadora, es necesario definir:

Dado que se busca una solución se precisan especificaciones de entrada y salida.

Para poder definir bien un problema es conveniente responder a las siguientes preguntas:

¿Qué entradas se requieren? (cantidad y tipo)
¿Cuál es la salida deseada? (cantidad y tipo)
¿Qué métodos y formulas se necesitan para procesar los datos?

Una recomendación muy práctica es el que nos pongamos en el lugar de la calculadora, y analizar que es necesario que me ordenen y en que secuencia, para poder producir los resultados esperados.

Analizado el problema, posiblemente tengamos varias formas de resolverlo; lo importante es determinar cual es la mejor alternativa: la que produce los resultados esperados en el menor tiempo y al menor costo. Claro que aquí también es muy válido el principio de que las cosas siempre se podrán hacer de una mejor forma.

11.3. Diseño del algoritmo

Una vez que sabemos cómo resolver el problema, pasamos a dibujar gráficamente la lógica de la alternativa seleccionada. Eso es precisamente un Diagrama de Flujo: la representación gráfica de una secuencia lógica de pasos a cumplir por la calculadora para producir un resultado esperado.

La experiencia nos ha demostrado que resulta muy útil trasladar esos pasos lógicos planteados en el diagrama a frases que indiquen lo mismo; es decir, hacer una codificación del programa pero utilizando instrucciones en español. Como si le estuviéramos hablando a la calculadora. Esto es lo que denominamos Pseudocódigo.

Cuando logremos habilidad para desarrollar programas, es posible que no elaboremos el diagrama de flujo; en su lugar podremos hacer directamente el pseudocódigo del programa

Las características de un buen algoritmo son:

- Debe poseer un punto particular de inicio.
- Debe ser definido, no debe permitir dobles interpretaciones.
- Debe ser flexible, soportando la mayoría de variantes que se puedan presentar en la definición del problema.
- Debe ser finito en tamaño y tiempo de ejecución.

11.3.1. Prueba de escritorio

Una vez escrito el algoritmo es necesario asegurarse de que éste realiza las tareas para las que ha sido diseñado, y que, por lo tanto, produce el resultado correcto y esperado. El modo más normal de comprobar un algoritmo es mediante su ejecución manual usando datos significativos que abarquen todo el posible rango de valores y anotando en una hoja de papel los valores que van tomando en las diferentes fases, los datos de entrada o auxiliares y, por último, los valores de los resultados. Este proceso se conoce como prueba del algoritmo o prueba de escritorio.

11.4. Codificación

Codificar (implementación del algoritmo) es escribir en lenguaje de programación de alto nivel la representación del algoritmo desarrollada en las etapas precedentes. Dado que el diseño de un algoritmo es independiente del lenguaje de programación utilizado para su implementación, el código puede ser escrito con igual facilidad en un lenguaje u otro. Para realizar la conversión del algoritmo en programa se deben sustituir las palabras reservadas en castellano por sus homónimos en inglés, y las operaciones e instrucciones indicadas en lenguaje natural expresarlas en el lenguaje de programación correspondiente.

11.5. Prueba y depuración

Los errores humanos dentro de la programación de calculadoras son muchos y aumentan considerablemente con la complejidad del problema. El proceso de identificar y eliminar errores, para dar paso a una solución sin errores se le llama depuración.

Al ejecutar un programa se pueden producir tres tipos de errores:

Errores de Compilación: Se producen normalmente por un uso incorrecto de las reglas del lenguaje de programación, suelen ser errores de sintaxis.

Errores de Ejecución: Se producen por instrucciones que la computadora puede comprender pero no ejecutar. En estos casos se detiene la ejecución del programa y se imprime un mensaje de error. Ejemplo de esto puede ser una división por cero.

Errores Lógicos: Se producen en la lógica del programa y la fuente del error suele ser el diseño del algoritmo, son más difíciles de detectar puesto que el programa puede funcionar y no producir errores de compilación ni de ejecución pero regresara resultados incorrectos. En este caso se debe regresar a la fase de diseño, modificar el algoritmo, cambiar el programa fuente y compilar y depurar una vez más.

11.6. Documentación

Es la guía o comunicación escrita en sus variadas formas, ya sea en enunciados, procedimientos, dibujos o diagramas.

A menudo un programa escrito por una persona, es usado por otra. Por ello la documentación sirve para ayudar a comprender o usar un programa o para facilitar futuras modificaciones (mantenimiento).

La documentación se divide en tres partes:

I. Documentación interna: Son los comentarios o mensaje que se añaden al código fuente para hacer más claro el entendimiento de un proceso.

II. Documentación externa: se define en un documento escrito los siguientes puntos:

- Descripción del problema (enunciado).
- Nombre del autor (Analista, programador).
- Algoritmo (Diagrama de flujo o pseudocódigo).
- Diccionario de Datos (Descripción de variables).
- Código fuente (programa).

III. Manual del Usuario: Describe paso a paso la manera como funciona el programa, con el fin de que el usuario obtenga el resultado deseado.

11.7. Mantenimiento

Se lleva a cabo después de terminado el programa, cuando se detecta que es necesario hacer algún cambio, ajuste o complementación al programa para que siga trabajando de manera correcta. Para poder realizar este trabajo se requiere que el programa este correctamente documentado.

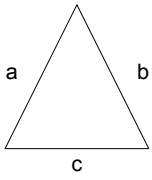
Ejemplos de algoritmos

Ejemplo 26: *Se desea un programa que determine el área de un triángulo.*

Definición del problema

Determinar el área de un triángulo, dado sus lados a b c

Análisis del problema



$$Area_{\Delta} = \sqrt{s(s-a)(s-b)(s-c)}$$

Donde:

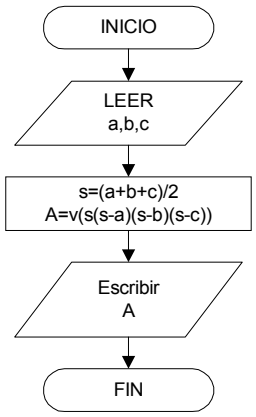
a, b, c = Son los lados del triángulo

$s = \frac{1}{2}(a+b+c)$ *(Semiperimetro según la formula de Heron)*

Datos de entrada: valores de los lados del triangulo (a, b, c)
Datos de salida: área del triángulo

Diseño del algoritmo

```
1  Inicio
2  Escribir "Area de un triangulo"
3  Leer a,b,c
4  Calcular s=(a+b+c)/2
5  Calcular A=√(s(s-a)(s-b)(s-c))
6  Escribir A
7  Fin
```



Prueba de escritorio

a	b	c	s	A
3	5	7	7.5	6.5
7	12	16	17.5	38.94

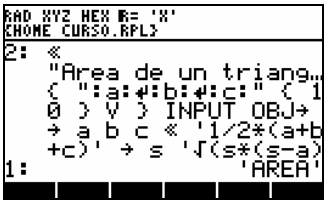
Codificación

```
« "Area de un triangulo"
{":a:
:b:
:c:" { 1 0 } V } INPUT OBJ→ → a b c
  « '1/2*(a+b+c)' → s
    '√(s*(s-a)*(s-b)*(s-c))' EVAL
  »
»
```

Prueba y depuración

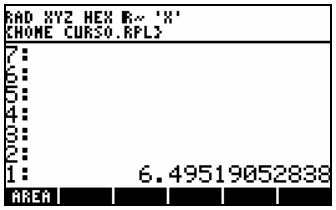
Escribimos el programa

```
« "Area de un triangulo"
{":a:
:b:
:c:" { 1 0 } V } INPUT OBJ→ → a b c
  « '1/2*(a+b+c)' → s
    '√(s*(s-a)*(s-b)*(s-c))' EVAL
  »
»
```



Guardamos el programa: 'AREA' STO

Probamos el programa: AREA
Ingresamos los valores de los lados del triangulo: 3 ▼5 ▼7 ENTER



Podemos probar el programa paso a paso con DEBUG y analizar el flujo de calculos de nuestro programa. El unico resultado es un número, podemos mejorar esta salida de datos, etiquetando este resultado con un nombre descriptivo, y redondeamos con dos cifras significativa el mismo.

Entonces, modificamos el codigo fuente, y nuestro codigo corregido será:

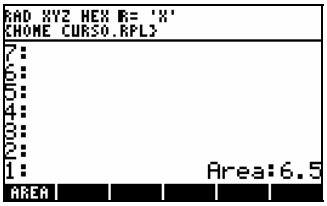
```
« "Area de un triangulo"
{":a:
:b:
:c:" { 1 0 } V } INPUT OBJ→ → a b c
  « '1/2*(a+b+c)' → s
    '√(s*(s-a)*(s-b)*(s-c))' EVAL
    2 RND :Area: →TAG
  »
»
```

Editamos el programa: ▶ EDIT ▼

Guardamos nuestro programa editado: ◀ EDIT

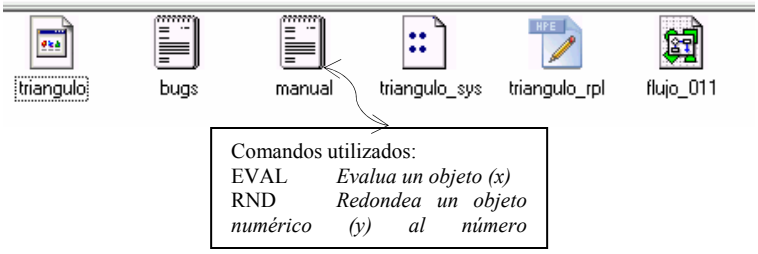
Ahora, ejecutamos nuestro programa: EDIT

Ingresamos los valores de los lados del triangulo: 3 ▼5 ▼7 ENTER



Excelente nuestro programa funciona sin problemas.

Documentación



Mantenimiento

Por ser muy espontáneo nuestro programa, no tenemos la necesidad de editar. Tal vez, podríamos modificar este programa en una nueva versión muy mejorada con aplicaciones de todos los elementos de un triangulo, como encontrar su altura, ángulos, perímetro, centro de gravedad, etc.

Ejemplo 27: *Usando datos recolectados de un experimento de laboratorio de física, del tiempo que oscila un péndulo simple de longitud L. Queremos calcular el valor de la aceleración de la gravedad en la ciudad de Oruro, El modelo matemático que rige nuestro péndulo es.*

$$P = 2 \cdot \pi \cdot \sqrt{\frac{L}{g}} \qquad (1)$$

Donde:
P = periodo de oscilación de un péndulo simple [s]
L = longitud del péndulo simple [m]
g = aceleración de la gravedad [m/s^2]

Se deben realizar los cálculos pertinentes, para llevar a las tablas correspondientes, siendo:

Linealización de la ecuación (1) para el trabajo experimental.
Valor del periodo de oscilación para cada ensayo.
Valores de las variables a ser graficadas.
Valores para ajustar las rectas.
Calcular la tangente del ángulo de inclinación de la recta obtenida en la grafica.
Calcular el valor de la aceleración de la gravedad.

Datos obtenidos en laboratorio.

Ensayo No.	**L** [cm]	No. de Oscilaciones	Tiempo **t** [s]
1	10	20	12,72
2	20	20	17,99
3	30	20	22,03
4	40	20	25,44
5	50	20	28,44
6	60	20	31,16
7	70	20	33,65
8	80	20	35,98
9	90	20	38,16
10	100	20	40,22

Definición del problema

Recalcamos, que no necesariamente debemos realizar un programa para resolver cada problema que nos plantean, debemos buscar algo generalizado mecánico que cumple cada problema, la mayoría se basa sobre un modelo matemático

Los objetivos de nuestro problema en general son:

Linealización de la ecuación (1) para el trabajo experimental.
Valor del periodo de oscilación para cada ensayo.
Valores de las variables a ser graficadas.
Valores para ajustar las rectas (ajuste de datos)
Calcular la tangente del ángulo de inclinación de la recta obtenida en la grafica.
Calcular el valor de la aceleración de la gravedad.

Todos estos problemas, podemos solucionarlo con un sencillo cálculo, lo que más nos va demorar es el ajuste de datos (ajuste de la curva), y esto es algo que utilizaremos en varios experimentos y/o practicas, entonces este será nuestro objetivo principal:

Diseñar un programa para la corrección de datos por el método de los mínimos cuadrados

Análisis del problema

Dado el modelo matemático $P = 2 \cdot \pi \cdot \sqrt{\frac{L}{g}}$

Y la respectiva linealización $P^2 = (\frac{4 \cdot \pi^2}{g}) \cdot L$

Las variables definidas, a, b = ?

$$\begin{array}{cccccc} Y & = & b & \cdot & X & + & a \\ \downarrow & & \downarrow & & \downarrow & & \downarrow \end{array}$$

$$P^2 = (\frac{4 \cdot \pi^2}{g}) \cdot L + 0$$

Variable de independiente L = X
Variable dependiente P^2 = Y

Método de los Mínimos Cuadrados.- aproxima el conjunto de puntos (x1,y1),(x1,y2)…(xN,yN) tiene por ecuación

$$Y = a + b \cdot X$$

Donde las constantes a y b quedan fijadas al resolver simultáneamente las ecuaciones:

$$\sum Y = a \cdot N + b \sum X$$
$$\sum X \cdot Y = a \cdot \sum X + b \sum X^2$$

Que se llaman ecuaciones normales para la recta de mínimos cuadrados.

Datos de entrada:

Datos variable independiente (L): como una lista {1 2 3….N}
Datos variable dependiente (P^2): como una lista {1 2 3…N}

Datos de salida:

$X^2=L^2$
$XY=L\cdot(P^2)$
(P^2)=Datos corregidos*
$\sum X$, $\sum Y$, $\sum XY$, $\sum X^2$, a, b

Diseño del algoritmo

```
 1 Inicio
 2 Escribir "AJUSTE DE DATOS MIN_2"
 3 Leer Datos Variable independiente X
 4 Leer Datos Variable dependiente Y
 5 @ Calcular
 6 XY = {x1 x2...xN}{y1 y2...yN}
 7 ∑XY = {xy1+xy2+...xyN}
 8 ∑X = {x1+x2...+xN}
 9 ∑Y = {y1+y2...+yN}
10 ∑X² = {(x1)²+(x2)²...+(xN)²}
11 ∑Y² = {(y1)²+(y2)²...+(yN)²}
12 [[A][B]]= [[ N ∑X ][ ∑X ∑X2 ]]⁻¹[[∑Y][ ∑XY]]
13 Yc = A + BX
14 Escribir "F(x)=a+bX",A,B
15 Escribir ∑X,∑Y,X²,Y²,∑X²,∑Y²,XY,∑XY,A,B,Yc
16 Fin
```

Codificación

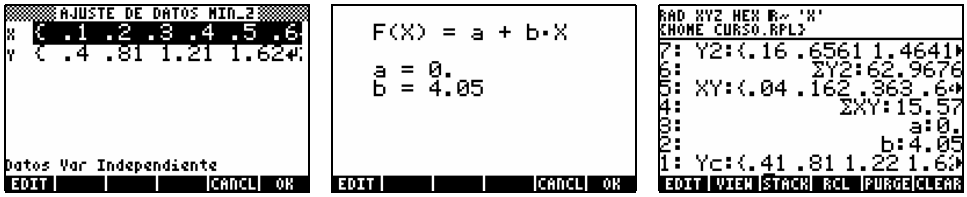
```
« "AJUSTE DE DATOS MIN_2"
 { { "X" "Datos Var Independiente" 5 }
 { "Y" "Datos Var Dependiente" 5 } }
 { 1 1 } { } { } INFORM DROP OBJ→ DROP
 SWAP 'X' STO 'Y' STO
 X Y * 'XY' STO XY Σ LIST 'SXY' STO
 X Σ LIST 'SX' STO Y Σ LIST 'SY' STO
 X SQ 'X2' STO X2 Σ LIST 'SX2' STO
 Y SQ 'Y2' STO Y2 Σ LIST 'SY2' STO
 X SIZE 'N' STO
 N SX SX SX2 { 2 2 } →ARRY INV
 SY SXY { 2 1 } →ARRY * ARRY→ DROP 2 RND
 'B' STO 2 RND 'A' STO
 B X * A ADD 'YC' STO N "N" →TAG
 SX "Σ X" →TAG SY "Σ Y" →TAG
 X2 "X2" →TAG SX2 "Σ X2" →TAG Y2 "Y2" →TAG
 SY2 "Σ Y2" →TAG XY "XY" →TAG SXY "Σ XY" →TAG
 A "a" →TAG B "b" →TAG YC 2 RND "Yc" →TAG
 CLLCD "   F(X) = a + b·X" 2 DISP
 "   a = " A →STR + 4 DISP
 "   b = " B →STR + 5 DISP 0 WAIT DROP
 { YC B A N SY2 Y2 SX2 X2 SY SX SXY XY Y X } PURGE
»
```

Prueba y depuración

Calculamos: $P = 2 \cdot \pi \sqrt{\frac{L}{g}}$

No.	L [m]	P [s]	L^2 [m^2]	P^2 [s^2]	$L \cdot P^2$ [ms^2]	P^2 (*) [s^2]	g [m/s^2]
1	0,10	0,64	0,01	0,40	0,04	0,41	9,85
2	0,20	0,90	0,04	0,81	0,16	0,81	9,75
3	0,30	1,10	0,09	1,21	0,36	1,22	9,77
4	0,40	1,27	0,16	1,62	0,65	1,62	9,75
5	0,50	1,42	0,25	2,02	1,01	2,03	9,77
6	0,60	1,56	0,36	2,43	1,46	2,43	9,75
7	0,70	1,68	0,49	2,83	1,98	2,84	9,76
8	0,80	1,78	0,64	3,24	2,59	3,24	9,75
9	0,90	1,91	0,81	3,64	3,28	3,65	9,76
10	1,00	2,01	1,00	4,04	4,04	4,05	9,77
Σ	5,50	14,27	3,85	22,24	15,57	22,24	97,68

*Datos corregidos



De la ecuación (1) $P^2 = \left(\frac{4 \cdot \pi^2}{g} \right) \cdot L + 0$ *donde:* $b = \frac{4 \cdot \pi^2}{g}$

Entonces: $g = \frac{4 \cdot \pi^2}{b}$ *y* $b = 4{,}05$ *(Pendiente de la recta)*

$$g = \frac{4 \cdot \pi^2}{4.05} = 9{,}7478 \qquad g = \frac{\sum_{i=1}^{N} g_i}{N} = \frac{97{,}68}{10} = 9{,}768$$

$$g = 9{,}7478 = 9{,}768$$

Entonces la gravedad en la ciudad de Oruro es 9,757 [m/s^2]

Capítulo III

Fundamentos de Programación

A pesar de las millones de aplicaciones inmersivas que circulan por todas partes, seria un error considerar la computadora como algo inteligente. Más bien, es una calculadora muy rápida con posibilidades de entrada de datos y visualización de los mismos. Si empezamos a disecar un ordenador, intentando examinar que pasa en su interior y como llega a realizar todo lo que hace, veremos que todo se reduce a:

Ejecutar operaciones aritméticas: El ordenador puede sumar, restar, multiplicar y dividir con mucha facilidad. Por extensión, puede realizar todo tipo de operaciones aritméticas.
Guardar valores: en un ordenador se pueden guardar valores (números, textos, etc.) en la memoria, y se pueden recuperar en cualquier momento. Si hace falta, se pueden también borrar valores de la memoria.
Comparar valores: se pueden comparar valores, y cambiar de camino de acción según el resultado de la comparación.
Ofrecer posibilidades de entrada de datos y salida o visualización de datos. Este es el canal que permite la interacción con el usuario humano.

Con estas posibilidades, tenemos herramientas suficientes para poder construir aplicaciones bien complejas. Sin embargo, no olvidemos que el ordenador no puede pensar por si mismo, ni entrar en nuestra cabeza. Si queremos llegar a un resultado determinado, tenemos que decirle con el mayor detalle, que es lo que tiene que hacer.

Programar es hacer que el ordenador obedezca una serie de instrucciones bien detalladas. Usando un lenguaje de programación, indicamos al ordenador que tiene que suceder en cada momento, y como tiene que reaccionar ante la interacción del usuario.

Se llama programar a la creación de un programa de computadora, un conjunto concreto de instrucciones que una computadora puede ejecutar. El programa se escribe en un lenguaje de programación, aunque también se pueda escribir directamente en lenguaje de máquina, con cierta dificultad. Entonces decimos que la programación de computadoras es el arte de hacer que una computadora haga lo que nosotros queramos.

En el nivel más simple consiste en ingresar en una computadora una secuencia de órdenes para lograr un cierto objetivo. Tal como le hablas a un amigo en un idioma. Para "hablarle" a una computadora es necesario utilizar un lenguaje en particular. El único lenguaje que una computadora entiende se denomina binario y tiene muchos dialectos. Desafortunadamente el lenguaje binario es muy difícil de leer y escribir para un humano, por lo cual debemos utilizar un lenguaje intermedio que después será traducido a binario. Esto es como una reunión en una cumbre entre nuestro presidente Morales y Lula presidente de Brasil. Morales habla, luego un intérprete repite en portugués lo que se ha dicho. Luego Lula contesta y un intérprete repite lo mismo en español.

Lo que traduce nuestro lenguaje intermediario a binario también se denomina intérprete. De la misma manera que es necesario disponer de un intérprete distinto para traducir del portugués al español que para hacerlo del árabe al ruso, será necesario disponer de un intérprete distinto para traducir las órdenes de RPL a binario y otra para traducir las de Basic.

12. User RPL

El User RPL se puede catalogar como un lenguaje de alto nivel (Los lenguajes de alto nivel son normalmente fáciles de aprender porque están formados por elementos de lenguajes naturales a nuestra comprensión humana), este es similar a muchos otro lenguajes que existen como Pascal, Visual Basic y otros. Además del User RPL (RPL del Usuario) existen el System RPL (RPL del sistema) y el ML (Machine Language: Lenguaje Máquina); algunas veces se le usa incorrectamente queriendo decir lenguaje ensamblador.

El User RPL basado en la manipulación de la pila; es un lenguaje estructurado tipo de programación que produce código con un flujo limpio, un diseño claro y un cierto grado de modularidad o de estructura jerárquica, dicho de otro modo cada programa tiene una entrada (principio del programa) y una salida (fin del programa). Se puede sacar provecho a la estructuración del lenguaje creando programas de bloque constitutivo; cada programa constitutivo puede permanecer solo o funcionar como una subrutina de un programa mayor: así, desde un programa se puede llamar a otro sin preocuparnos de nada ya que una vez finalizado el programa llamado se devolverá el flujo al programa principal. A pesar de ser lento, es muy poderoso y fácil de usar, lo único que se debe saber es algo de algoritmia, lógica y aprender los comandos de codificación (sintaxis de programación). Los comandos son muy buenos, con ellos puede hacerse prácticamente todo lo imaginable: manejar cadenas y gráficos (incluso sprites, muy útil para los juegos), realizar operaciones de cualquier tipo y acceder a la infinidad de funciones preprogramadas de que dispone la calculadora.

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

12.1. Orígenes del User RPL

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

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

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

Se tuvieron en cuenta varios lenguajes y sistemas operativos ya existentes pero ninguno cumplía con todos los objetivos del diseño; por consiguiente se desarrolló un nuevo sistema, el cual mezcla la interpretación entrelazada de Forth con el enfoque funcional de Lisp. El sistema operativo resultante, conocido de modo no oficial como RPL (Reverse-Polish Lisp), hizo su primera aparición pública en junio de 1986 en la calculadora Business Consultant 18C. Más tarde, RPL ha sido la base de las calculadoras HP-17B, HP-19B, HP-27S, HP-28C y HP-28S y HP 48S y HP 48SX. La HP-17B, 18C y la 19B se diseñaron para aplicaciones de negocios; ellas y la calculadora científica HP-27S ofrecen una lógica de cálculo algebraica y el sistema operativo subyacente es invisible para el usuario. Las familias HP 28/HP 48 de calculadoras científicas usan una lógica RPN y muchas de la facilidades del sistema operativo están disponibles directamente como comandos.

Los objetivos oficiales del sistema operativo listados anteriormente se fueron mezclando a lo largo del ciclo de desarrollo del RPL con un objetivo menos formal de creación de un lenguaje de control matemático que extendería la facilidad de uso y la naturaleza interactiva de una calculadora al reino de las operaciones de la matemática simbólica. En este contexto, una calculadora se distingue de un ordenador por:

- Tamaño muy compacto;
- "encendido instantáneo" sin calentamiento o carga de software.
- Teclas dedicadas para las operaciones comunes.
- "acción instantánea" cuando se pulsa una tecla de función.

13. Un programa en User RPL

Un programa es en general una secuencia de comandos, es decir todo el proceso para resolver un determinado problema, pasó a paso, lógicamente.

Un programa para nuestra calculadora es nada más que un objeto definido por los delimitadores « », este contiene un conjunto de instrucciones, números u objetos, que se desea ejecutar en forma automática para realizar una tarea.

Por ejemplo, si queremos realizar un programa que sume dos números enteros, nuestro programa será: « 2 3 + » o también sin cambiar el programa, podríamos mostrarlo con un comando u objeto por línea, similar a otros lenguajes de programación:

```
«          Inicio del programa
 2 3       ...Objetos...
 +         ...Comandos...
»          Final del programa
```

Ejemplo 28: *Programa que calcula la suma de dos números*

Paso 1. Escribir el programa: ← − 2 SPC 3 + ENTER
Paso 2. Nombre del programa: α ◀ 0 1 ENTER
Paso 3. Guardar el programa: STO

Paso 1. Escribir el programa: → + 2 SPC 3 + ENTER
Paso 2. Nombre del programa: ALPHA P 0 1 ENTER
Paso 3. Guardar el programa: STO▸

Bien hemos creado nuestro primer programa. Para almacenar este programa en el directorio del usuario, en el menú VAR, se ha guardado como cualquier otro objeto.

Para ejecutar el programa se puede colocar en la pila y ejecutar con EVAL

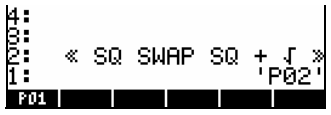
O llamarlo por su nombre, buscamos la tecla correspondiente al programa VAR [programa] el resultado se devuelve en la pila.

Ejemplo 29: *Programa que halla la hipotenusa de un triangulo rectángulo dado los valores de los catetos; estos tendrán que estar en la pila.*

Código fuente	Comentarios
«	*Inicio del programa*
SQ	*Eleva al cuadrado el numero del nivel 1*
SWAP	*Invierte los objetos de la pila del nivel 1 y 2*
SQ	*Eleva al cuadrado el numero del nivel 1*
+	*Suma los números del nivel 1 y 2*
√	*Halla la raíz cuadrada del numero del nivel 1*
»	*Fin del programa*

Paso 1. Escribir el programa: ← − ← √x ← ▶ ← √x + √x ENTER
Paso 2. Nombre del programa: α ◀ 0 2 ENTER
Paso 3. Guardar el programa: STO

Paso 1. Escribir el programa:
→ + ← √x ALPHA S W A P ALPHA ← √x + √x ENTER
Paso 2. Nombre del programa: ALPHA P 0 2 ENTER
Paso 3. Guardar el programa: STO▶



Ejecutamos el programa: P02

Este programa lo hubiéramos hecho también de esta forma:

```
«                 Inicio del programa
 →V2              Combina los numero del nivel 1 y 2 para formar un
                  vector bidimensional o un número complejo
 ABS              Halla el valor absoluto
»                 Fin del programa
```

Ejecutamos el programa: P02A

En este ejemplo mostramos que podemos optimizar nuestro código en pocas líneas y así usar menos memoria, también se ejecutara más rápido, solo tenemos que aprovechar de las funciones preprogramadas que tiene nuestra calculadora.

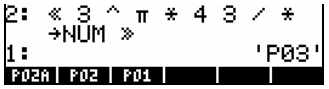
Ejemplo 30: *Dado el radio, hallar el volumen de una esfera.*

Formula que nos permite hallar el volumen de una esfera: $Vol_{esfera} = \frac{4}{3}\pi r^3$

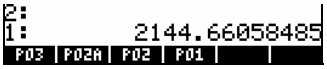
Código fuente	Comentarios
`«`	*Inicio del programa*
`3 ^`	*Eleva al cubo el numero del nivel 1*
`π`	*Coloca en la pila a en el nivel 1*
`*`	*Halla el producto de los números del nivel 1 y 2*
`4 3 /`	*Coloca en la pila el cociente de estos do números*
`*`	*Halla el producto de los números del nivel 1 y 2*
`→NUM`	*Convierte el objeto simbólico a numérico*
`»`	*Fin del programa*

Paso 1. Escribir el programa:
↩ − 3 y^x ↩ SPC × 4 SPC 3 ÷ × ↩ EVAL ENTER
Paso 2. Nombre del programa: α ◀ 0 3 ENTER
Paso 3. Guardar el programa: STO

Paso 1. Escribir el programa:
↱ + 3 y^x ↩ SPC × 4 SPC 3 ÷ × ↱ ENTER ENTER
Paso 2. Nombre del programa: ALPHA P 0 3 ENTER
Paso 3. Guardar el programa: STO▸



Ejecutamos el programa [P03]



13.1. Ejecutar un programa paso a paso, para corregir errores

Podemos verificar si existe algún error en el procedimiento de nuestro programa, como, ejecutando un programa paso a paso, para esto seguimos los siguientes pasos:

1. Ponga todos lo datos requeridos por el programa en la pila en los niveles apropiados.
2. Ponga el programa o el nombre del programa en el nivel 1 de la pila.
3. Vamos a menú RUN: PRG NXT [MODES] [DEBUG] para empezar e inmediatamente suspenda la ejecución. El indicador HALT se visualizara en el área de estado.
4. Realice la acción que estime oportuna:
   - Para ver el siguiente paso del programa visualizado en el área de estado y posteriormente ejecutado, pulse [SST]
   - Para visualizar, pero no ejecutar el paso o los dos pasos siguientes del programa, pulse [SST↓]
   - Para continuar con la ejecución normal, pulse ↩ CONT
   - Para abandonar otra ejecución, pulse [KILL]
5. Repita el paso 4 cuando Ud. requiera hacerlo.

Ejemplo 31: *Programa que calcula el área total y volumen de un cilindro, dados el radio y la altura, estos tendrán que estar en la pila.*

$$Area_{cilindro} = 2 \cdot \pi \cdot (r^2 + r \cdot h) \qquad Volumen_{cilindro} = \pi \cdot r^2 \cdot h$$

```
« DUP2 SWAP DUP SQ 3 ROLLD
* + 2 π * * →NUM "Area" →TAG 3 ROLLD
SWAP SQ * π * →NUM "Vol" →TAG
»
```

Ejecutamos el programa paso a paso:

1. Colocamos en la pila los valores del radio y la altura del cilindro

3 SPC 7 ENTER

3 SPC 7 ENTER

2. Colocamos el nombre del programa en la pila

' [alpha] ENTER

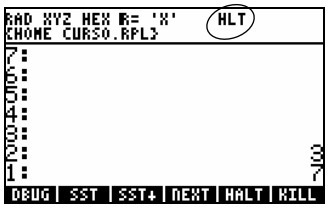
' [alpha] ENTER

3. Vamos al menú RUN

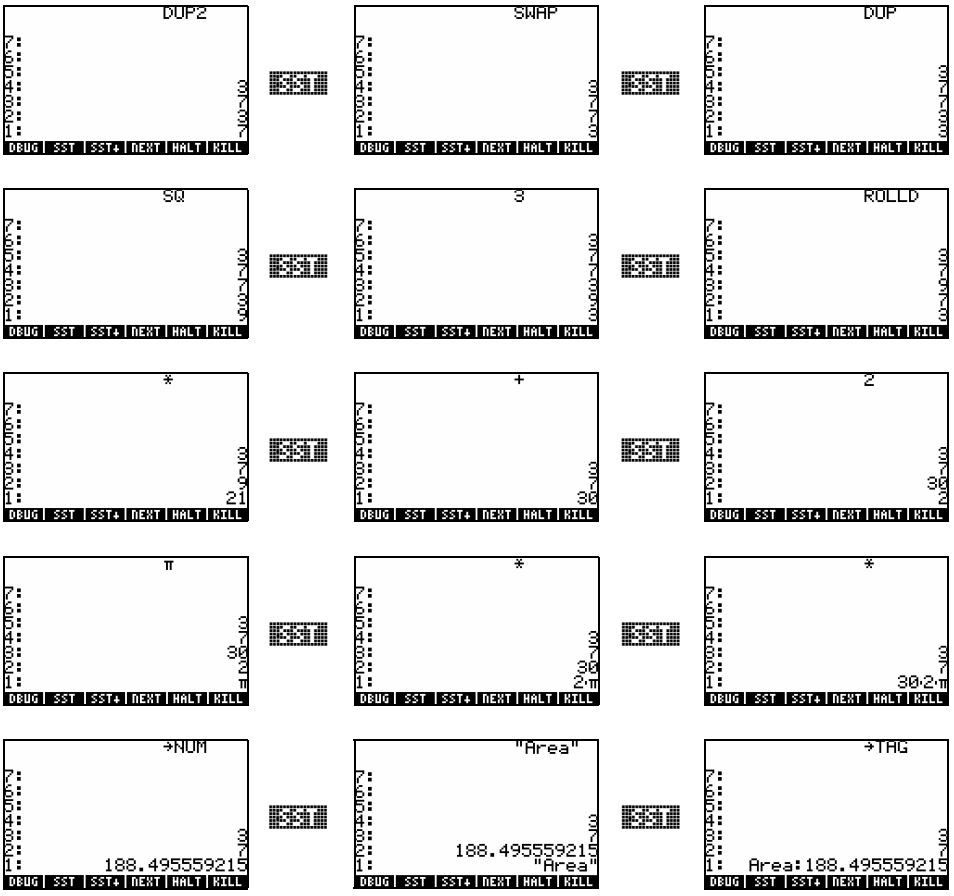
41 MENU: PRG NXT [RUN]

41 MENU: ↰ PRG NXT NXT [RUN]

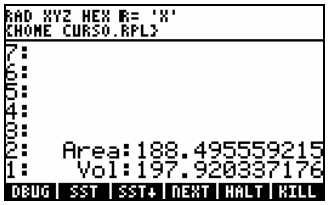
4. Presionamos ▒▒▒ *se suspende la ejecución del programa e inmediatamente aparece el indicador* HALT *en el área de estado.*



5. Para ver el siguiente paso del programa visualizado en el área de estado y posteriormente ejecutarlo, pulsamos ▒▒▒ *ó* ▒▒▒



Si queremos que el programa continué normalmente, pulsamos ↩ CONT



13.2. Ejecutar un programa paso a paso desde la mitad

Para realizar la operación paso a paso desde la mitad de un programa seguimos los siguientes pasos:

1. Introduzca un comando HALT en el programa en el lugar en que desee empezar el funcionamiento paso a paso.
2. Ejecute el programa con normalidad. El programa se detiene al ejecutarse el comando HALT y se visualiza el indicador HALT
3. Realice la acción que estime oportuna:
   - Para ver el siguiente paso del programa visualizado en el área de estado y posteriormente ejecutado, pulse SST
   - Para visualizar, pero no ejecutar el paso o los dos pasos siguientes del programa, pulse NEXT
   - Para continuar con la ejecución normal, pulse ↩ CONT
   - Para abandonar otra ejecución, pulse KILL
4. Repita el paso 3 cuando desee hacerlo.
5. Cuando desee que el programa vuelva a ejecutarse normalmente, retire el comando HALT del programa.

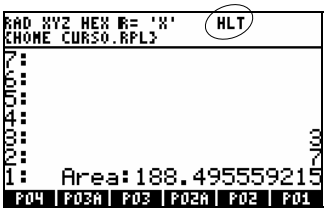
Ejemplo 32: *Ejecutar paso a paso desde la mitad, el programa P04 (ejemplo 29).*

Editamos el Prg.: → PRG ▼

Editamos el Prg: → PRG NXT ▼

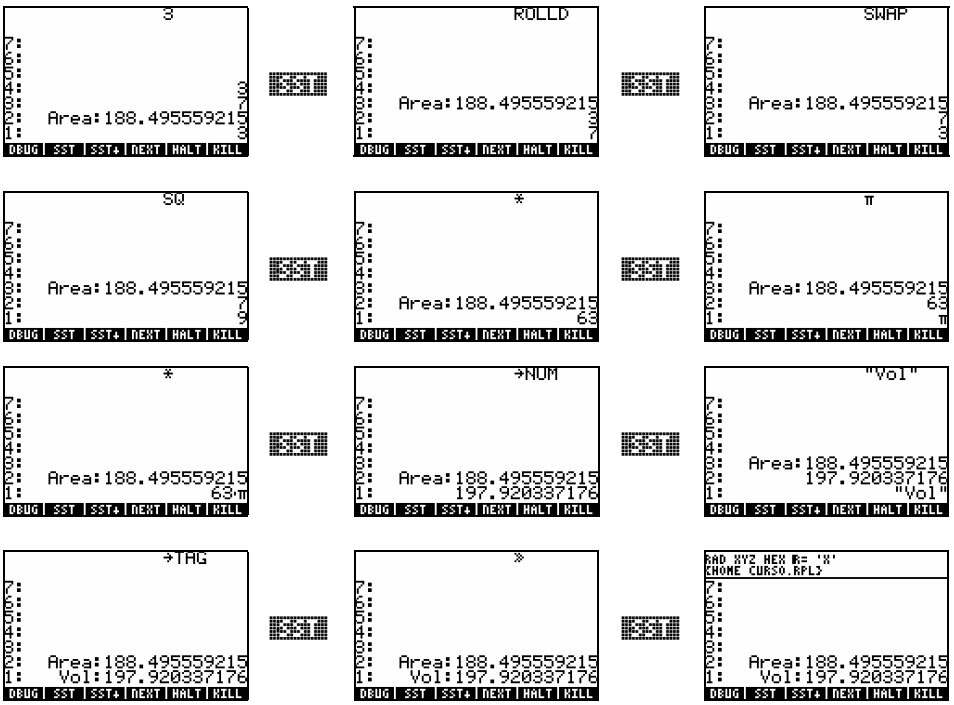
Guardamos el Prg.: ↶ ENTER *Guardamos Prg:* ↶ ENTER

Ejecutamos el programa VAR ENTER *(Colocar en la pila, el valor del radio y la altura):*



El programa se ha detenido a la mitad en su ejecución, y aparece el indicador HALT *para ver el siguiente paso del programa visualizado en el área de estado y posteriormente ejecutarlo, pulsamos* SST *ó* SST↓

41 MENU: PRG NXT MORE SST *41 MENU:* ↶ PRG NXT NXT MORE SST



Si existen subrutinas. Para ejecutar la subrutina en un paso, pulse SST
Para ejecutar la subrutina paso a paso SST↓

14. Declaración de variables

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

14.1. Variables Locales

Las variables locales son aquellas variables temporales dentro de los programas, es decir solo están presentes en la ejecución de los programas. Es muy recomendable el uso de estas variables ya que utiliza menos memoria y son más rápidos al ejecutarse.
El comando → define el nombre de las variables y la asignación respectivamente. Para declarar variables locales se sigue la siguiente sintaxis:

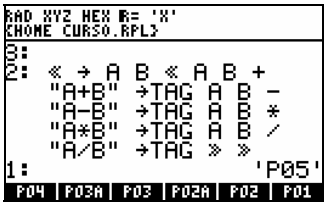
```
« valor1 valor2...valorN → nombre1 nombre2...nombreN
  'Operación algebraica' »

« valor1 valor2...valorN → nombre1 nombre2...nombreN
  « ...programa... » »
```

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

Ejemplo 33: *Programa para calcular la suma, resta, multiplicación y división de dos números, puesto en la pila.*

Código fuente	Comentarios
`«`	*Inicio del programa*
`→ A B`	*Declaración de variables locales de A y B*
	→ recoge dos valores de la pila y los almacena en las variables definidas.
`«`	*Inicio de subprograma*
`A B + "A+B" →TAG`	*Realiza la operación respectiva de A y B*
`A B - "A-B" →TAG`	*el comando →TAG etiqueta el objeto (resultado)*
`A B * "A*B" →TAG`	*con un nombre descriptivo cada uno*
`A B / "A/B" →TAG`	*respectivamente según la operación.*
`»`	*Fin del subprograma*
`»`	*Fin del programa*

Escribimos el programa y lo guardamos '`P05`' STO



Ejecutamos el programa: VAR [ ] *(Necesita dos números puestos en la pila)*

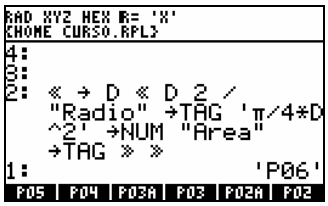
El comando (→) es el encargado de recoger 2 valores de la pila y almacenarlos en las variables definidas: El nivel 2 se almacena en A, el nivel 1 en B. En seguida se ha iniciado otro programa o subprograma, esto se hará generalmente cuando se tengan programas complejos cuya secuencia de comandos sea mucho más extensa, aclarando que las variables declaradas solo son validas en este subprograma.

Ejemplo 34: *Programa que calcula el área y el radio de un circulo, dado el diámetro.*

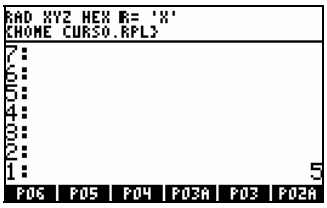
Escribimos el programa:

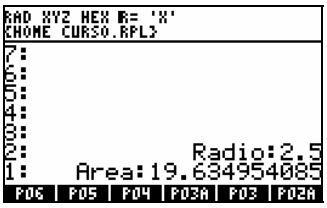
```
« → D
  « D 2 / "Radio" →TAG
    'π/4*D^2' →NUM "Area" →TAG
  »
»
```

Guardamos el programa: '`P06`' STO



Antes de ejecutar necesitamos un valor en la pila.
Ingresamos el valor del diámetro: 5 ENTER
Ejecutamos el programa: VAR [:]





Ejemplo 35: *Programa que calcula el área de un triángulo, dados las longitudes de sus lados.*

$$Area_{\Delta} = \sqrt{s(s-a)(s-b)(s-c)}$$

Donde:

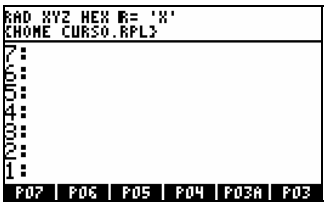
a, b, c = Son los lados del triángulo

$s = \frac{1}{2}(a+b+c)$ *(Según la formula de Heron)*

Escribimos el programa:

```
« → a b c
  « '1/2*(a+b+c)' → s
    '√(s*(s-a)*(s-b)*(s-c))'
  »
»
```

Guardamos el programa: 'P07' STO



Ingresamos los valores de los lados del triangulo: 3 SPC 5 SPC 7 ENTER
Ejecutamos el programa: [:]

Ejemplo 36: *Dado a, b, c, d, u y v puestos en la pila, resolver el siguiente sistema en notación RPN.*

$$ax + by = u$$
$$cx + dy = v$$

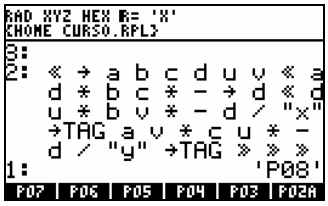
Donde:

$$x = \frac{du - bv}{ad - bc} \qquad y = \frac{av - cu}{ad - bc}$$

Escribimos el programa:


```
« → a b c d u v
  « a d * b c * - → d
    « d u * b v * - d /
      a v * c u * - d /
    »
  »
»
```

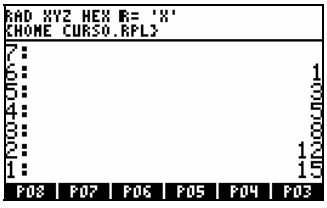
Guardamos el programa: '`P08`' STO

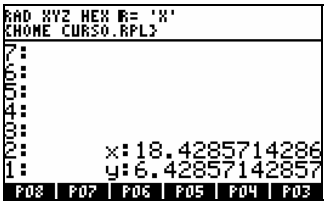


Ingresamos valores para a, b, c, d, u v:

1 SPC 3 SPC 5 SPC 8 SPC 1 2 SPC 1 5 ENTER

Ejecutamos el programa: ▮20:▮





14.2. Variables globales

Las variables globales son todos los objetos que se tienen almacenados en la calculadora y las que se van a crear de manera permanente (por decirlo de alguna forma) manualmente ó por medio de los programas. Su uso dentro de los programas representa una desventaja, ya que el trabajo con estas variables es un poco lento.

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

La sintaxis a seguir para crear una variable local desde un programa es:

```
« valor1 'nombre1' STO
  valor2 'nombre2' STO
  .
  .
  .
  valorN 'nombreN' STO

»
```

Alcance de las variables globales: Una variable global es accesible al directorio donde se define y a cualquier subdirectorio unido a ese directorio, a menos que una variable con el mismo nombre exista en un subdirectorio bajo consideración. Las consecuencias de esta regla son las siguientes:

- Una variable global definida en el directorio HOME será accesible de cualquier directorio dentro de HOME, a menos que esté redefinida dentro de un directorio o un subdirectorio.
- Si usted redefine la variable dentro de un directorio o de un subdirectorio esta definición toma precedencia sobre cualquier otra definición en directorios sobre el actual.

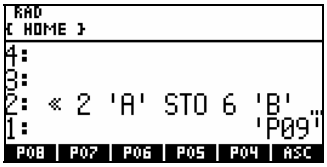
Ejemplo 37: *Realizar un programa que evalué A^2+B^2*

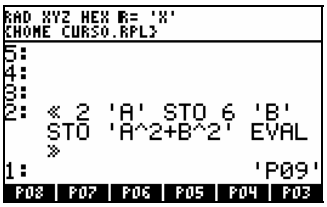
Escribimos el programa:



```
« 2 'A' STO
  6 'B' STO
  'A^2+B^2' EVAL
»
```

Guardamos el programa: '`P09`' STO▶





Ejecutamos el programa: VAR ▒2:▒

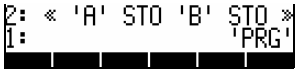
Al ejecutar el programa notaremos que han aparecido dos nuevas variables en el directorio actual. Si deseamos el valor de estas, solo la llamamos por su nombre ▒1▒ y ▒3▒ respectivamente. Estas variables pueden ser utilizadas por cualquier programa.

Ejemplo 38: *Programa para calcular la suma, resta, multiplicación y división de dos números.*

Escribimos el programa:

```
«
  4 'A' STO   5 'B' STO
  A "A" →TAG B "B" →TAG
  'A+B' EVAL DUP 'a' STO "A+B" →TAG
  a A - "(A+B)-A" →TAG A B * DUP 'b' STO
  "A*B" →TAG b A / "(A*B)/A" →TAG
»
```

Los valores de A y B no necesariamente deben estar en el programa; estos pueden ser recogidos de la pila. Ejemplo:

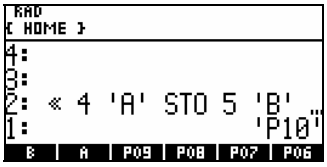


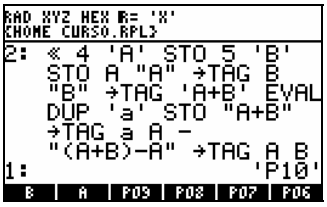
Guardamos el programa: STO▶

Colocamos dos valores en la pila: 2 SPC 3 ENTER

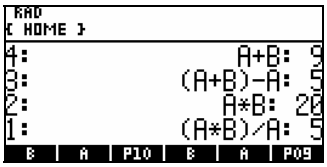
Ejecutamos el programa: ▒2:▒
Se han creado las variable A y B

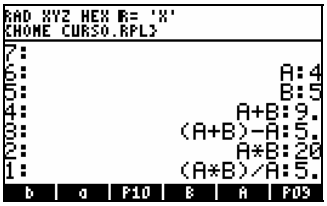
Guardamos el programa: `'P10'` STO





Ejecutamos el programa: VAR P10





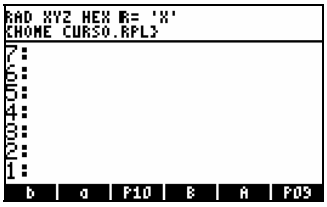
Nótese que se han creado dos variables nuevas con los mismos nombres.

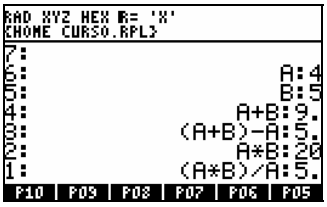
User RPL es sensible a la diferencia entre mayúsculas y minúsculas, por ejemplo: Tiempo, tiempo y TIEMPO representan tres variables distintas. Asegúrese de escoger nombres que le ayuden a recordar lo que se esta almacenando en las variables. Lo mismo sucede con los comandos; todos los comandos de programación deberán estar en mayúsculas.

Si queremos que las variables no aparezcan tendríamos hacer uso del comando `PURGE` para borrarlas. Modificamos nuestro programa ingresando al final del programa `{ b a B A } PURGE` como se borra un conjunto cualquiera de objetos.

```
«
  4 'A' STO
  5 'B' STO
  A "A" →TAG B "B" →TAG
  'A+B' EVAL DUP 'a' STO "A+B" →TAG
  a A - "(A+B)-A" →TAG
  A B * DUP 'b' STO "A*B" →TAG
  b A / "(A*B)/A" →TAG
  { b a B A } PURGE
»
```

Ejecutamos el programa: VAR





Las variables globales que anteriormente hemos creado con nuestro programa quedan borradas.

Ejemplo 39: *Programa que resuelve el siguiente sistema de ecuaciones*

$$\begin{aligned} 2x+2y+9z &= 12 \\ 4x+6y+6z &= 18 \\ 8x+12+14 &= 32 \end{aligned} \qquad \begin{bmatrix} 2 & 2 & 9 \\ 4 & 6 & 6 \\ 8 & 12 & 14 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 12 \\ 18 \\ 32 \end{bmatrix}$$

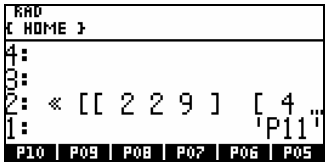
Resolvemos mediante una ecuación de matriz simple de la forma A·X=B

$$X = A^{-1} \cdot B$$

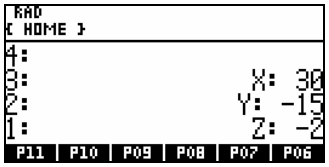
Escribimos el programa:


```
«
  [ [ 2 2 9 ]
  [ 4 6 6  ]
  [ 8 12 14 ] ] 'A' STO
  [[12][18][32]] 'B' STO
  A INV B * OBJ→ DROP 3 →LIST
  « "X" "Y" "Z"  » →TAG OBJ→ DROP
  « B A » PURGE
»
```

Guardamos el programa: 'P11' STO



Ejecutamos el programa: VAR P12



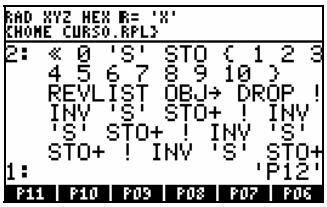
Ejemplo 40: *Realice un programa para calcular la suma de la siguiente serie:*

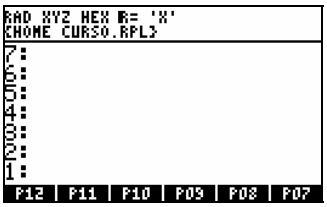
$$S = \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + ... + \frac{1}{10!}$$

Escribimos el programa:

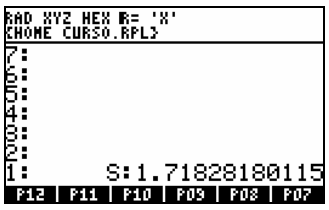
```
«
  0 'S' STO
  { 1 2 3 4 5 6 7 8 9 10 }
  REVLIST OBJ→ DROP
  ! INV 'S' STO+ ! INV 'S' STO+ ! INV 'S' STO+
  ! INV 'S' STO+ ! INV 'S' STO+ ! INV 'S' STO+
  ! INV 'S' STO+ ! INV 'S' STO+ ! INV 'S' STO+
  ! INV 'S' STO+ S "S" →TAG
»
```

Guardamos el programa: 'P12' STO





Ejecutamos el programa: VAR P12



El en ejemplo 40 estamos haciendo uso del comando STO+
Cuando trabajamos con variables globales normalmente se usa los siguientes comandos:

STO	Almacena el objeto (x) en la variable (y)
STO+	Suma un número u otro objeto al contenido de una variable especificada.
STO-	Calcula la diferencia entre el contenido de una variable y el número u otro objeto, y almacena en una variable especificada.
STO*	Multiplica el contenido de una variable especificada mediante un número u otro objeto
STO/	Calcula el cociente del contenido de una variable especificada y un número u otro objeto especificado, y almacena el resultado en una variable especificada.
RCL	Recupera el objeto almacenado en una variable especificada (x) en la pila.

14.3. Variables locales compiladas

Las variables locales compiladas son una gran opción en comparación de las variables globales y locales, ya que las variables globales ocupan mucha memoria, y las variables locales no pueden usarse fuera del programa en que fueron creados. Entonces aquí trabajan las variables locales compiladas, su uso depende de la necesidad y requerimientos de programación. Las variables locales compiladas pueden ser usadas en subprogramas o subrutinas que se ejecutan bajo el programa principal, solamente hay que llamarlas por su nombre (previa declaración) como a la subrutina en el programa principal, cuando el programa termina la variable ya no existe

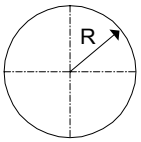
El comando ← define el nombre de las variables locales compiladas. Para declarar variables locales compiladas se sigue la siguiente sintaxis:

```
« valor1 valor2...valorN → ←nombre1 ←nombre2 ... ←nombreN
  « nombresubrutina
    ←nombre1 ←nombre2 ... ←nombreN
  »
»
```

nombresubrutina:

```
« ←nombre1 ←nombre2 ... ←nombreN »
```

Ejemplo 41: *Dado el radio de un círculo, calcular el área y el perímetro del mismo.*



$$A = \pi \cdot r^2$$

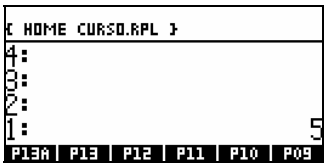
$$P = 2 \cdot \pi \cdot r$$

Programa principal: P13

```
« → ←R
  « P013A
    2 π * ←R * →NUM 2 RND
    "Perimetro" →TAG
  »
»
```

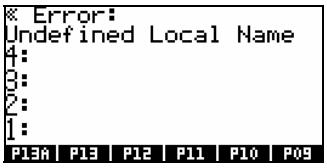
Subrutina: P13A

```
« ←R SQ π * →NUM 2 RND
  "Area" →TAG
»
```



En este ejemplo mostramos la declaración de la variable local compilada ←R *que es la manera como se le llamara (antecedido del signo* ←*) en otros subprogramas o subrutinas. Pero solo las subrutinas llamadas por el programa principal.*

Nótese que en la subrutina ya no es necesario declarar la variable R, porque ya fue declara en el programa principal. Si ejecutamos solamente el programa P13A *(Subrutina* P13A*), sin llamarlo desde cualquier programa, nos botara un mensaje de error. Ejecutamos*



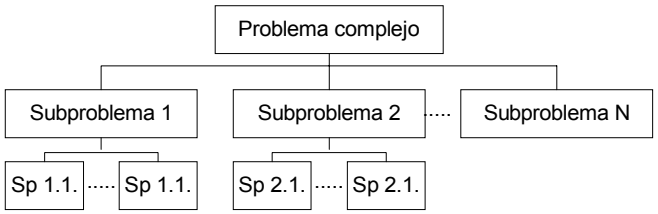
Uno de los métodos más conocidos para resolver un problema es dividirlo en problemas más pequeños, llamados subproblemas. De esta manera, en lugar de resolver una tarea compleja y tediosa, resolvemos otras más sencillas y a partir de ellas llegamos a la solución. Esta técnica se usa mucho en programación ya que programar no es más que resolver problemas, y se le suele llamar diseño descendente, metodología del divide y vencerás o programación top-down.

Es evidente que si esta metodología nos lleva a tratar con subproblemas, entonces también tengamos la necesidad de poder crear y trabajar con subprogramas para resolverlos. A estos subprogramas se les suele llamar módulos, de ahí viene el nombre de programación modular. En User RPL disponemos de las subrutinas o módulos (Conjunto de subprogramas relacionados)

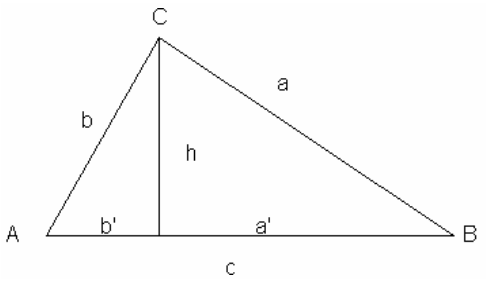
Un subprograma, es un segmento de código que recibe unos datos llamados objetos de entrada, a partir de los cuales realiza una serie de operaciones para devolver resultados mediante los objetos de salida.

Módulo, es un conjunto de subrutinas, funciones, definiciones de datos derivados, variables globales, se presenta como un algoritmo separado del algoritmo principal, el cual permite resolver una tarea específica y puede ser llamado o invocado desde el algoritmo principal cuando sea necesario. los módulos son independientes en el sentido de que ningún módulo puede tener acceso directo a cualquier otro módulo.

Si tenemos proyectos muy complejos (grandes) podemos crear subproblemas (subrutinas), como se muestra en la figura:



Ejemplo 42: *Calcular el área, la altura y el perímetro de un triángulo, dado las longitudes de sus lados.*



$$Area_{\Delta} = \sqrt{s(s-a)(s-b)(s-c)}$$

Donde: a, b, c = lados del triángulo

$2p = a + b + c$ *(Perímetro)*

$s = \frac{1}{2}(a+b+c)$ *(Semiperimetro)*

$h = \frac{2 \cdot A}{c}$ *(Altura)*

Programa principal: P14

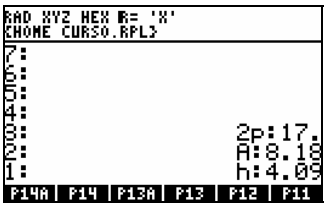
```
« → ←a ←b ←c
  « P14A »
»
```

Subrutina: P14A

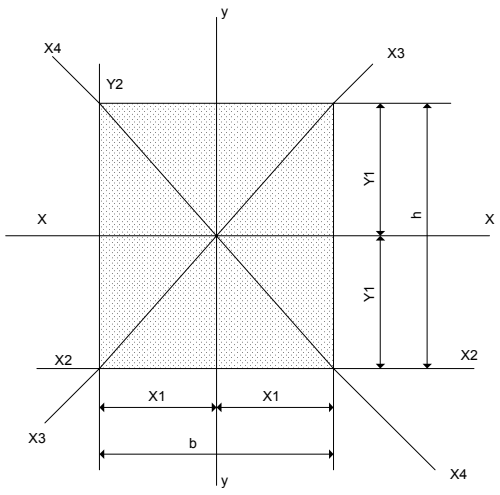
```
« ←a ←b + ←c +
  DUP 2 RND "2p" →TAG
  SWAP 2 / → s
  '√(s*(s-←a)*(s-←b)*(s-←c))'
  DUP 2 RND "A" →TAG
  SWAP 2 * ←c / 2 RND "h" →TAG
»
```

Ejecutamos el programa: [P14]

Teniendo cuidado en el ingreso de datos, cumpliéndose: $a > b - c$ *y* $a < b + c$ *para las formulas.*



Ejemplo 43: *Calcular todas las características geométricas de un rectángulo, dado las longitudes de sus lados.*



Área:

$$A = b \cdot h$$

Perímetro:

$$P = 2 \cdot a + 2 \cdot h$$

Centroides:

$$X_1 = \frac{b}{2} \qquad Y_1 = \frac{h}{2}$$

Momentos de Inercia:

$$I_{x1} = \frac{Ah^2}{12} \qquad I_{y1} = \frac{Ab^2}{12}$$

$$I_{x2} = \frac{Ah^2}{3} \qquad I_{y2} = \frac{Ab^2}{3}$$

$$I_{x2y2} = \frac{b^2 \cdot h^2}{4}$$

$$I_{x3} = I_{x4} = \frac{b^2 \cdot h^2}{6 \cdot \left(b^2 + h^2\right)}$$

Momento polar de inercia:

$$J_p = \frac{b \cdot h}{12} \cdot \left(b^2 + h^2\right)$$

Módulos de la sección, axiales:

$$W_x = \frac{A \cdot h}{6} \qquad W_y = \frac{A \cdot b}{6}$$

Programa principal: P15

```
« → ←b ←h
  « ←b ←h * → ←a
    « ←b "b" →TAG ←h "h" →TAG
      ←a "A" →TAG P15A P15B
    »
  »
»
```

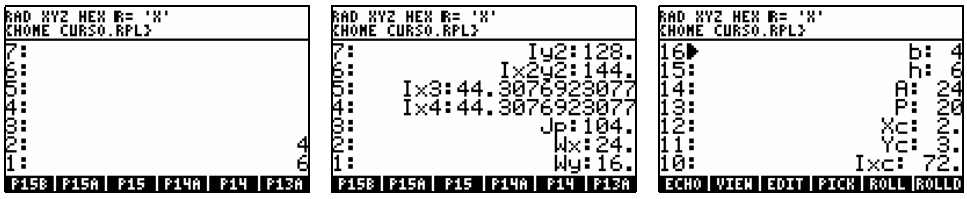
Subrutina: P15A

```
« ←b ←h + 2 * "P" →TAG
  ←b 2 / "Xc" →TAG
  ←h 2 / "Yc" →TAG
»
```

Subrutina: P15B

```
« ←h SQ ←a * 12 / "Ixc" →TAG
  ←b SQ ←a * 12 / "Iyc" →TAG
  ←h SQ ←a * 3 / "Ix2" →TAG
  ←b SQ ←a * 3 / "Iy2" →TAG
  ←b SQ ←h SQ * 4 / "Ix2y2" →TAG
  ←b 3 ^ ←h 3 ^ * ←b SQ ←h SQ + 6 * /
  DUP "Ix3" →TAG SWAP "Ix4" →TAG
  ←b ←h * 12 / ←b SQ ←h SQ + * "Jp" →TAG
  ←a ←h * 6 / "Wx" →TAG
  ←a ←b * 6 / "Wy" →TAG
»
```

Ejecutamos el programa: *Ingresamos los valores de la base y la altura en la pila, b y h respectivamente.*

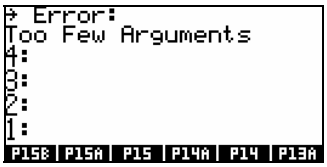


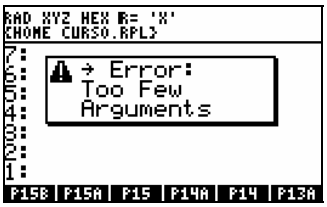
Podemos reasignar (dar un nuevo valor) o agregar valores adicionales al valor de una variable, según sea necesario, ejemplos:

La variable v1 que en principio tenia un valor de cero con la asignación de un nuevo valor, v1 tiene otro valor.

15. Entrada de datos

Si podemos ver los ejemplos anteriores, en la mayoría, se han recogido valores de la pila, si no se encuentran valores o cualquier otro objeto en la pila el programa botara un mensaje de error: `Too Few Arguments`





El cual nos indica que necesita argumentos para operar o recoger valores para nuestras variables. Frecuentemente se utiliza la pila para realizar este proceso, pero existe métodos más eficiente, la calculadora dispone de comandos especiales para esta tarea, además permite darle una buena presentación a nuestros programas.

15.1. INPUT

Input permite usar el área de la pila para recoger datos, es un editor similar a la línea de comandos, permite proporcionar entradas predefinidas, y le permite al usuario usar funciones desde teclado.

Input crea un formulario simple para recoger datos, mostrando una línea de comandos donde se puede encontrar vacía o de lo contrario con una cadena de caracteres (etiqueta) que identifique a una variable para su respectiva asignación de valor de esa variable.

Para crear esta plantilla de entrada de datos se sigue la siguiente sintaxis:

```
« "cadena" "cadenalineaCmdo" INPUT OBJ→ »
```

O también:

```
« "cadena" { "cadenalineaCmdo" { fila columna } modo(s) }
  INPUT OBJ→
»
```

"cadena"	Texto que aparecerá como encabezado
"cadenalineaCmdo"	Texto que aparecerá en la línea de comandos
{ #fila #columna }	Ubicación del cursor en la línea de comandos. Por defecto es { 1 0 }
Modo(s)	Será el modo por defecto, que presentará la línea de comandos.
INPUT	Suspende la ejecución del programa. Muestra el mensaje ecadenam en la parte superior de la pila e indica que se introduzcan datos en la línea de comandos.

Modos:

ALG	Activa el modo algebraico para introducir objetos algebraicos
α	Activa alpha para introducir una cadena de caracteres.
V	Para realizar una comprobación de la línea de comandos, verifica si existe algún error de sintaxis.
-1	Cambia el modo de entrada de la línea de comandos de modo insertar (cursor ▮) al modo sustituir (cursor ▒) y viceversa.

Para tener acceso a los comandos de entrada de datos presionamos: PRG NXT
También podemos ir a este menú escribiendo en la pila: 39 MENU

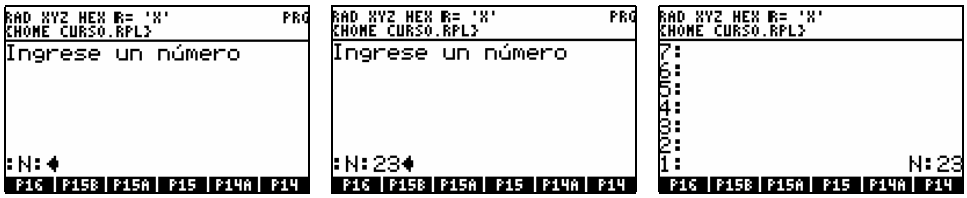
Ejemplo 44: *Programa para recoger datos.*

Escribimos el programa P16

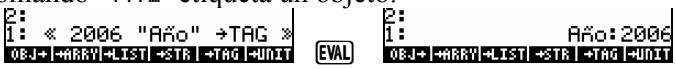
```
«                          Inicio del programa
 "Ingrese un número"       Coloca el titulo como encabezado
  ":N:"                    Etiqueta con el nombre N al objeto ingresado
  INPUT                    Detiene el programa para recoger el objeto
  OBJ→                     Descompone el objeto ":N:" en :N:Objeto
»                          Fin del programa
```

Guardamos el programa: 'P16' STO

Ejecutamos el programa: ▓▓▓
Ingresamos un valor cualquiera: 2 3 ENTER



Un objeto etiquetado es aquel que tiene una etiqueta, que nada más es un nombre descriptivo que se le da a un objeto.
En el ejemplo No. 44 estamos etiquetando el objeto que vamos a ingresar en la línea de comandos con la etiqueta (nombre) N.
El comando →TAG etiqueta un objeto:



El comando DTAG quita la etiqueta de un objeto:

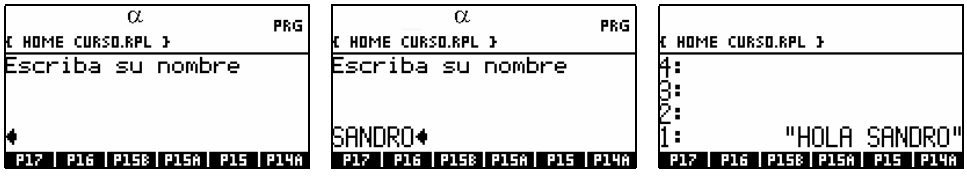
Otro ejemplo: :NombreEtiqueta:ValorObjeto

Ejemplo 45: *Programa para pedir datos de cadena de caracteres (alfabéticos)*

Escribimos el programa P17

```
« "Escriba su nombre"
  { "" { 1 0 } α } INPUT → n
  « "HOLA " n + »
»
```

Guardamos el programa: 'P17' STO *Ejecutamos:* ▓▓▓



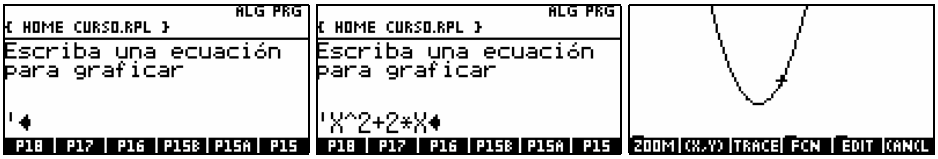
Nótese en el ejemplo que se ha activado el modo alpha (α)

Ejemplo 46: *Realizar un programa para graficar una ecuación.*

Escribimos el programa P18

```
« "Escriba una ecuación
para graficar"
  { "'" { 1 0 } ALG }
  INPUT OBJ→
  STEQ ERASE DRAW PICTURE
  { PPAR EQ } PURGE
»
```

Guardamos el programa: 'P18' STO *Ejecutamos:* ▦21:▦



Ejemplo 47: *Realizar un programa para hallar la media aritmética de una serie de datos.*

Escribimos el programa P19

```
«                          Inicio del programa
"Cáculo de x̄               Coloca el titulo como encabezado
Ingrese los datos"
  { "{}"                   Cadena que aparece en la línea de comandos
  { 1 2 }                  Coloca el curso en la fila uno y la columna 2
   V }                     Verifica si existe algún error en la línea de Cmdos
  INPUT                    Detiene el programa para el ingreso de datos
  OBJ→                     Descompone el objeto "{}" en {}
  DUP                      Duplica el objeto del nivel 1
  Σ LIST                   Realiza la sumatoria de todos los objetos de la
                           lista del nivel 1
  SWAP                     Invierte los objetos del nivel 1 y 2
  SIZE                     Determina el tamaño de la lista del nivel 1
  /                        Calcula el cociente de los objetos del nivel 1 y 2
  "x̄" →TAG                 Etiqueta el objeto del nivel 1 con el nombre x̄
»                          Fin de programa
```

Guardamos el programa: `'P19'` STO *Ejecutamos:* [P19]

Ejemplo 48: *Dado dos vectores calcular el producto vectorial de estos.*

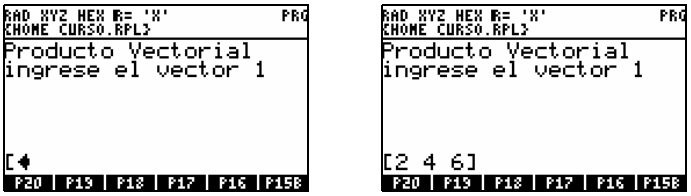
$$\begin{bmatrix}2 & 4 & 6\end{bmatrix}\times\begin{bmatrix}2 & 6 & 3\end{bmatrix}=\begin{bmatrix}i & j & k\\ 2 & 4 & 6\\ 2 & 6 & 3\end{bmatrix}=(12-36)i-(6-12)j+(12-8)k$$

$$v=-24i+6j+4k$$

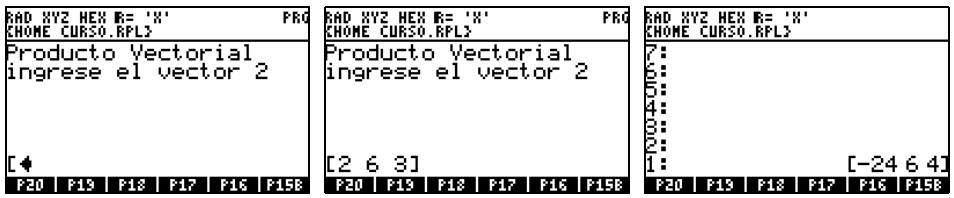
Escribimos el programa `P20`

```
« "Producto Vectorial
ingrese el vector 1"
{ "[]" {1 2} V } INPUT OBJ→
"Producto Vectorial
ingrese el vector 2"
{ "[]" {1 2} V } INPUT OBJ→ → v1 v2
« v1 v2 CROSS »
»
```

Ejecutamos: [P20] *ingresamos el primer vector:* 2 SPC 4 SPC 6 ENTER



Ingresamos el segundo vector: 2 SPC 6 SPC 3 ENTER



Ejemplo 49: *Realizar un programa para calcular el porcentaje de humedad en los agregados finos (arena) y grueso (grava).*

Expresión para determinar el porcentaje de Humedad:

$$\%W = \frac{P_{mi} - P_{mf}}{P_{Pmi}} \times 100$$

Procedimiento experimental.

a. *Se elige el banco de material a analizarse.*
b. *Se saca la capa seca superficial de éste, capa que por acción del sol, viento, etc., esta mas seca y no refleja la humedad de la mayoría del material que yace debajo.*
c. *Habiendo llegado a la capa húmeda, cuartear.*
d. *Se pesa la muestra cuarteada y se obtiene Pmi = peso de la muestra inicial.*
e. *Se seca el material, con una hornilla o un horno a 110 ºC por un tiempo que sea necesario. Y se vuelve a pesar y obtenemos Pmf = peso de la muestra final.*

Escribimos el programa P21

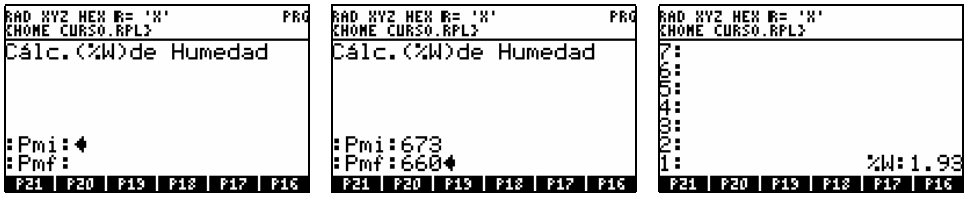
```
« "Cálc.(%W)de Humedad"
{ ":Pmi:
:Pmf:" ( 1 0 ) V )
  INPUT OBJ→ DTAG
  SWAP DTAG SWAP → Pmi Pmf
  « Pmi Pmf - Pmi / 100 *
    2 RND
    "%W" →TAG
  »
»
```

Guardamos el programa: 'P19' STO

Ejecutamos: [▒▒] *e ingresamos los datos recolectados:*

Arena 1: Pmi = 72.0; Pmf = 69.0: 7 2 ▼ 6 9 ENTER

Grava 1: Pmi = 673.0; Pmf = 660.0: 6 7 3 ▼6 6 0 ENTER



En la siguiente tabla presentamos los datos calculados:

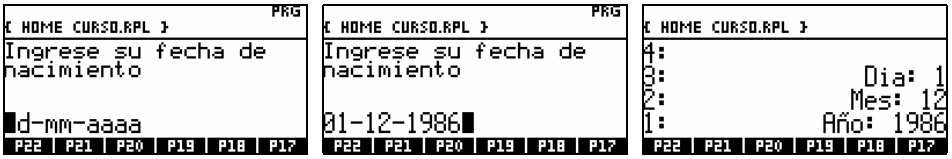
Grupo	Pmi	Pmf	%W
Arena 1	*72.0*	*69.0*	*4.17*
Arena 2	*70.0*	*67.0*	*4.28*
Grava 1	*673.0*	*660.0*	*1.93*
Grava 2	*345.0*	*338.0*	*2.03*

Ejemplo 50: *Realizar un programa para ingresar por la línea de comandos el día, mes y año, del nacimiento de una persona.*

Programa P22

```
« "Ingrese su fecha de
nacimiento"
{ "dd-mm-aaaa" -1 }
 INPUT → dma
 « dma 1 2 SUB OBJ→ "Dia" →TAG
   dma 4 5 SUB OBJ→ "Mes" →TAG
   dma 7 10 SUB OBJ→ "Año" →TAG
 »
»
```

Ejecutamos: ▓P22▓
Ingresamos los datos: 0 1 ▶ 1 2 ▶ 1 9 8 6 ENTER



El comando SUB *extrae la parte de una lista, cadena o matriz, o un objeto de gráficos especificado (z) definida por las posiciones inicial (y) y final (x)*

15.2. INFORM

Este comando permite crear formularios para la entrada de datos, este devuelve dos valores en la pila, en el nivel 1 un valor de `1` si el formulario fue llenado ó `0` si se ha cancelado el uso del formulario o por cualquier error, en el nivel 2 una lista con los objetos recogidos de los campos del formulario. Para crear un formulario con INFORM se sigue la siguiente sintaxis:

```
« "Titulo"
 { {"Etiqueta" "Cadena de Ayuda" TipodeObj(s) } }
 { #columna #espacio } { reconfiguracion } { pordefecto }
 INFORM
»
```

Donde:

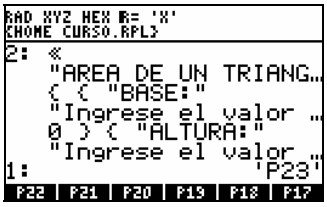
`"Titulo"`	Titulo del formulario
`"Etiqueta"`	Texto (etiqueta) descriptivo del campo
`"Cadena de Ayuda"`	Texto de ayuda del campo
`TipodeObj(s)`	Tipo de objeto(s) que el campo puede aceptar
`{ #columna #espacio }`	Número de columnas y el número de espacio de los campos. Por defecto para un campo es `{ 1 0 }`
`{reconfiguracion}`	Valores de reconfiguración del campo (cuanto hacemos uso del menú RESET). Por defecto es `{ }`
`{ pordefecto }`	Valor por defecto que tiene los campos. Por defecto es `{ }`
`INFORM`	Muestra en pantalla una plantilla de entrada definida con los anteriores argumentos.

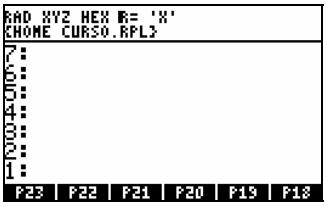
Para tener acceso a los comandos de entrada de datos presionamos: ↰ PRG NXT ▦ También podemos ir a este menú escribiendo en la pila: `39 MENU`

Ejemplo 51: *Realizar un programa para hallar el Área de un triángulo dados la base y la altura.* `P23`

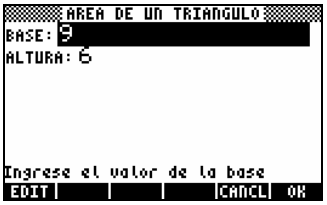
```
« "AREA DE UN TRIANGULO"
  { { "BASE:" "Ingrese el valor de la base" 0 }
  { "ALTURA:" "Ingrese el valor de la altura" 0 } }
  { 1 0 } { 0  0 } { 9 6 } INFORM DROP OBJ→ DROP → B H
  « B "B" →TAG H "H" →TAG
    B H * 2 / "A" →TAG
  »
»
```

Guardamos el programa P23



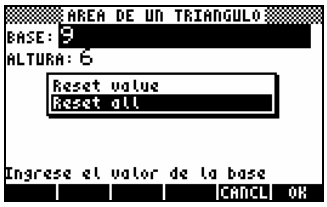


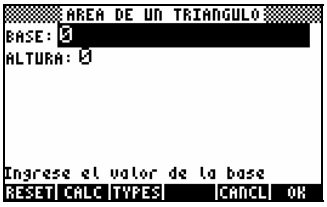
Ejecutamos: [P23]



Por defecto, se muestran valores en los campos de entrada; en nuestro programa hemos designamos estos valores con { 9 6 }
Ver código P23
Nótese el texto de ayuda de los campos se muestra en la parte baja del formulario.

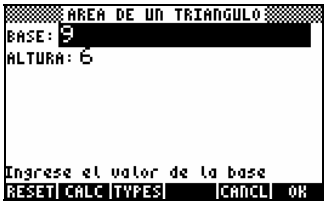
Podemos reconfigurar estos valores: [NXT] [RESET] [▼] [0]

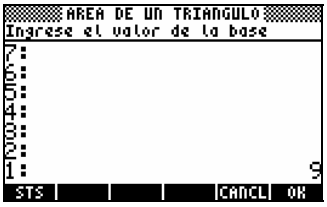




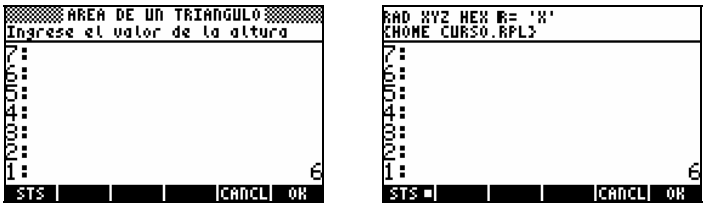
En los campos de entrada se muestra los valores de reconfiguración; en nuestro programa hemos designamos esto con { 0 0 } *Ver código* P23

Con [EVAL] *podemos copiar el contenido del campo actual a la pila y muestra la pila en pantalla, Se utiliza para efectuar operaciones de cálculo paralelo o para ir a otras partes de la calculadora mientras se trabaja dentro de una plantilla de entrada.*



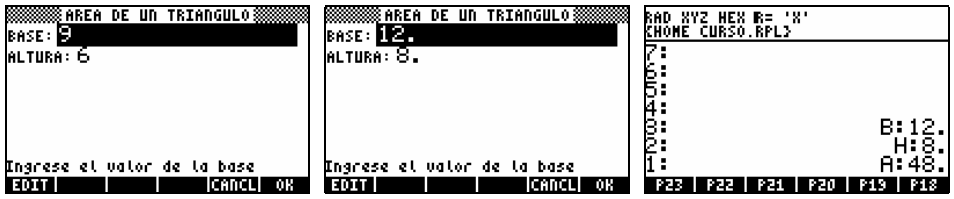


visualiza la línea de estado con el directorio actual, para continuar presionamos el mismo



Podemos realizar cualquier operación. Para continuar presionamos

Para terminar el programa P23 *ingresamos datos en los campos:*
1 2 OK 8 OK OK

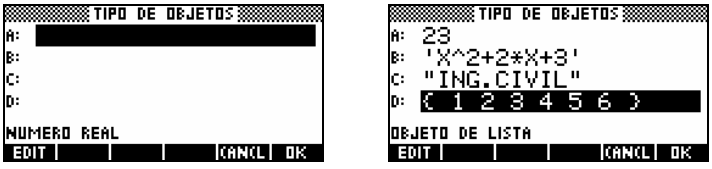


Ejemplo 52: *Ejemplo para mostrar como podemos restringir el tipo de objetos de un campo.*

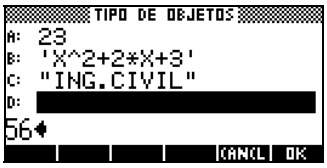
P24

```
« "TIPO DE OBJETOS"
  { { "A:" "NUMERO REAL" 0 }
  { "B:" "OBJETO ALGEBRAICO" 9 }
  { "C:" "OBJETO DE CADENA, STRING" 2 }
  { "D:" "OBJETO DE LISTA" 5 }
  } { 1 1 }
  { } DUP
  INFORM DROP
»
```

0 *Numero Real*
9 *Exp. Algebraica*
2 *Cadena*
5 *Lista*

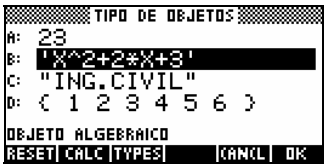


Si no escribimos el tipo de objeto de un cambo que indica el mensaje de ayuda del mismo, nos botara un mensaje de error:

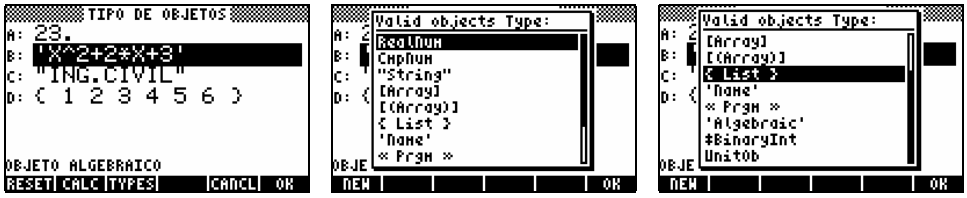


No se puede ingresar un objeto (Tipo 0) número real, el campo D: solo acepta objetos de lista (tipo 5). Para ver el tipo de objetos véase la Pág. 12.

Con TYPES *podemos saber que tipo de objetos que acepta un campo de entrada.*



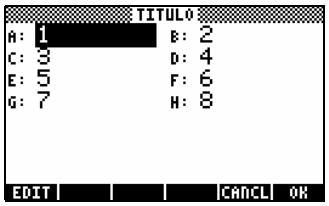
Si no restringiríamos con `2` *en el código del programa* `24`*, nuestro campo aceptaría los tipos de objetos mostrados en el cuadro:*



Con NEW *podemos escribir el objeto seleccionado:*

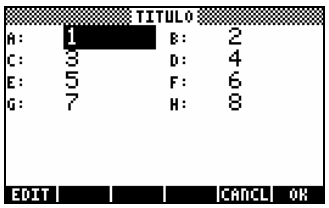
Ejemplo 53: *Ejemplo para mostrar el formato de un campo de entrada*

```
P25

« "TITULO"
  { "A:" "B:" "C:" "D:" "E:" "F:" "G:" "H:" }
  { 2 1 } {1 2 3 4 5 6 7 8 } DUP
  INFORM DROP
»
```



Formato: { 2 1 } *dos columnas un espacio de tabulación para el campo.*

```
P25A
```

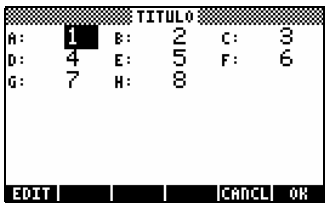


Formato: { 2 4 } *dos columnas cuatro espacios de tabulación para el campo.*

```
P25B
```

Formato: { 3 2 } *tres columnas dos espacios de tabulación para el campo.*

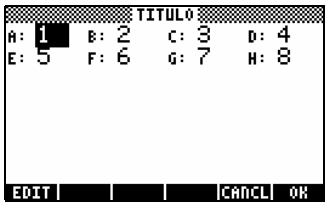
```
P25C
```



Formato: { 3 4 } *tres columnas cuatro espacios de tabulación para el campo.*

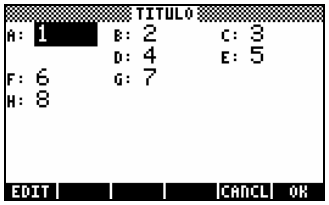
P25D

Formato: ≪ 4 2 ≫ *cuatro columnas dos espacios de tabulación para el campo.*



P25E

```
« "TITULO"
  { "A:" "B:" "C:" { } "D:" "E:" "F:" "G:" { } "H:" }
  { 3 2 } {1 2 3 4 5 6 7 8 } DUP INFORM DROP
»
```



Formato: ≪ 3 2 ≫ *tres columnas dos espacios de tabulación para el campo.*
Con una lista vacía ≪ ≫ *también se puede presentar campos sin formato.*

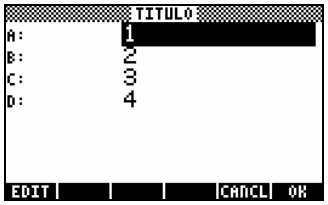
P25F

```
« "TITULO"
  { "A:" "B:" "C:" "D:" { } { } { }
  { } "E:" "F:" "G:" "H:" } { 4 2 }
  {1 2 3 4 5 6 7 8 } DUP INFORM DROP
»
```

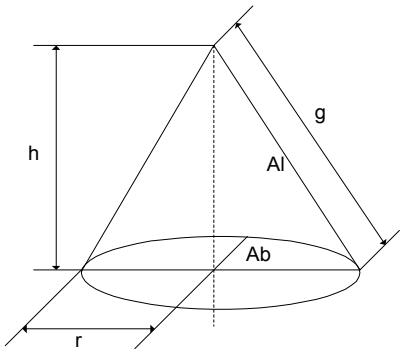
Formato: ≪ 4 2 ≫ *cuatro columnas dos espacios de tabulación para el campo.*

P25G

Formato: ≪ 1 8 ≫ *una columna ocho espacios de tabulación para el campo.*



Ejemplo 54: *Realizar un programa para hallar el área y el volumen de un cono circular recto, dados el radio y la altura.*



Volumen:

$$V = \frac{\pi \cdot r^2 \cdot h}{3}$$

Generatriz:

$$g = \sqrt{h^2 + r^2}$$

Área lateral:

$$Al = \pi \cdot r \cdot g$$

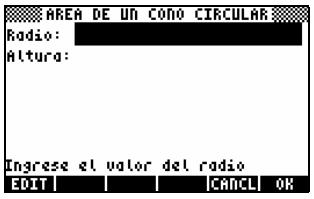
Área total:

$$Ab = \pi \cdot r^2 + Al$$

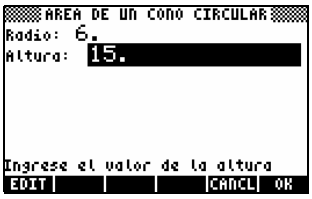
P26

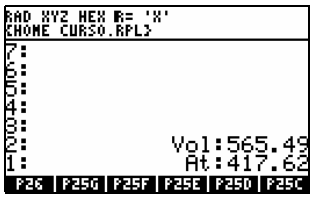
```
« "AREA DE UN CONO CIRCULAR"
  { { "Radio:" "Ingrese el valor del radio" 0 }
  { "Altura:" "Ingrese el valor de la altura" 0 } }
  { 1 1 } { } DUP INFORM DROP OBJ→ DROP → r h
  « r SQ h * π * 3 / →NUM 2 RND "Vol" →TAG
    h SQ r SQ + √ → g 'π*r*g' →NUM → Al
    'π*r^2+Al' →NUM 2 RND "At" →TAG
  »
»
```

Ejecutamos el programa: [P2:]



Ingresamos valores: [6] [.] [1] [5] [.] [.]





Ejemplo 55: *Realizar un programa para hallar los parámetros hidráulicos A, P, R y T para el canal de sección trapezoidal que se muestra en la figura.*

Área hidráulica:

$$A = (b + z \cdot y)y$$

Perímetro mojado:

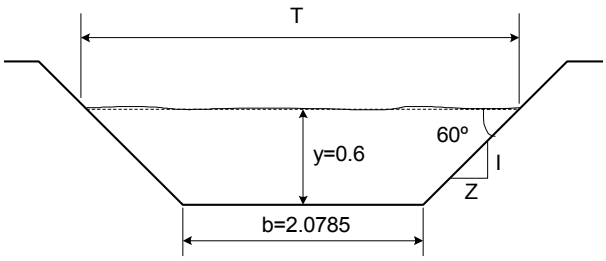
$$P = b + 2 \cdot y\sqrt{1 + z^2}$$

Radio hidráulico:

$$R = \frac{(b + z \cdot y)y}{b + 2 \cdot y\sqrt{1 + z^2}}$$

Ancho superficial:

$$T = b + 2 \cdot z \cdot y$$

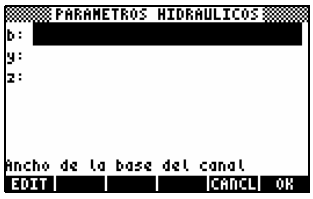


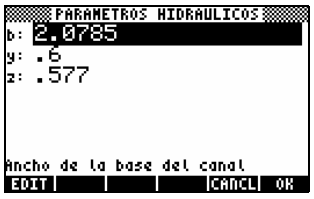
P27

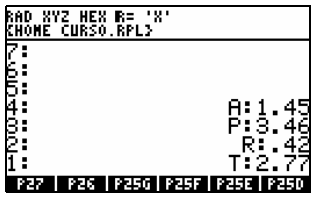
```
« "PARAMETROS HIDRAULICOS"
  { { "b:" "Ancho de la base del canal" }
  { "y:" "profundidad del agua en el canal" }
  { "z:" "Talud de las paredes del canal" } }
  { 1 1 } { } DUP
  INFORM DROP OBJ→ DROP → b y z
  « y z * b + y * → A
    « A "A" →TAG
      'b+2*y*√(1+z^2)' →NUM 2 RND DUP
      "P" →TAG SWAP → P
      'A/P' EVAL 2 RND "R" →TAG
      b z y * 2 * +  "T" →TAG
    »
  »
»
```

$$z = ctg60^\circ = \frac{1}{\sqrt{3}} = \frac{\sqrt{3}}{3}$$

Ingresamos valores: 2 . 0 7 8 5 OK
0 . 6 OK 0 . 5 7 7 OK OK







Ejemplo 56: *Realizar un programa para hallar la media aritmética, desviación media, varianza, el valor máximo y mínimo; dado los siguientes datos:*

4.78	*4.56*	*4.89*	*4.60*	*4.90*	*437*	*4.56*	*4.76*	*4.73*	*4.65*	*4.71*	*4.45*

P28

```
« "ANALISIS DE DATOS"
  { { "X:" "Datos Registrados como lista {}" 5 } }
  { 1 1 } { } { } INFORM DROP OBJ→
  DROP OBJ→ →ARRY OBJ→ 1 + →ARRY STOΣ
  "X = " MEAN 2 RND + " ± " + SDEV 2 RND +
  { { "ΣX:" "Suma de datos" }
  { "N:" "Numero de Datos" }
  { "X̄:" "Media aritemtica" }
  { "MD:" "Desviación Standart" }
  { "Max:" "Valor maximo" }
  { "Min:" "Valor minimo" }
  { "s:" "Varianza" } } { 2 1 }
  ΣX NΣ MEAN 2 RND SDEV 2 RND MAXΣ MINΣ VAR 2 RND 7 →LIST
  DUP INFORM DROP { ΣPAR ΣDAT } PURGE
»
```

Comandos utilizados:

→ARRY	*Combina los números en un sistema*
STOΣ	*Almacena la matriz de estadística actual aΣDAT*
ΣX	*Devuelve la suma de los datos deΣDAT*
NΣ	*Devuelve el número de filas deΣDAT*
MEAN	*Calcula la media de los datos estadísticos de ΣDAT*
SDEV	*Calcula la desviación estándar de la columna deΣDAT*
MAXΣ	*Encuentra el valor máximo de la matriz de Est.ΣDAT*
MINΣ	*Encuentra el valor mínimo de la matriz de Est.ΣDAT*
VAR	*Calcula la variación de los datos deΣDAT*
→LIST	*Combina los objetos entre el nivel 1 y el nivel actual en una lista*

Ejecutamos el programa: [▦] e ingresamos los datos en una lista:

15.3. CHOOSE

Este comando muestra una lista de opciones en un pequeño cuadro por las cuales podemos desplazarnos y elegir la opción deseada, si de este es seleccionada alguna opción devuelve 1 (verdadero) y en el nivel 2 el objeto seleccionado, si no se selecciona ninguna opción devuelve 0 (falso) indicando que se ha cancelado la operación. Para crear esta lista de opciones se sigue la siguiente sintaxis:

```
« "Titulo"
  { { "NombreItem1" Obj1 }
  { "NombreItem2" Obj2 }
  { "NombreItem3" Obj3 }
  .
  .
  { "NombreItemN" ObjN } }
  #Posicion  CHOOSE
»
```

Donde:

"Titulo"	Titulo del cuadro de selección
"NombreItemN"	Objeto que se muestra en el cuadro
ObjN	Objeto que será devuelto si se selecciona un ItemN. Este puede ser un número para enumerar un Item.
#Posicion	Número que indica la posición del la selección en el cuadro por defecto es 1
CHOOSE	Muestra en pantalla un cuadro de selección definido con los anteriores argumentos.

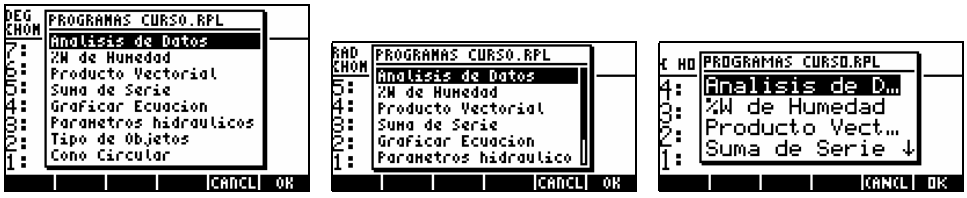
Para tener acceso a CHOOSE desde la calculadora presionamos: ↰ PRG NXT ▦ También podemos ir a este menú escribiendo en la pila: `39 MENU`

Ejemplo 57: *Ejemplo para mostrar los diferentes formatos de posición.* `P29`

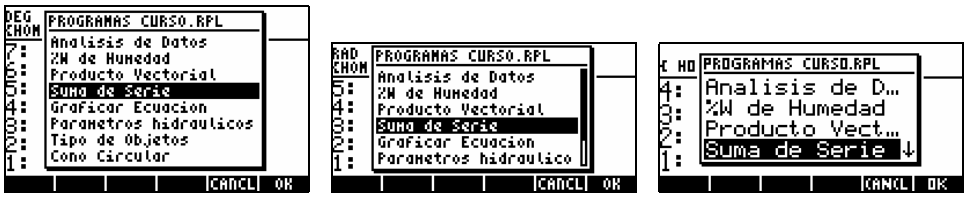
```
« "PROGRAMAS CURSO.RPL"
  { { "Analisis de Datos" P28 }
  { "%W de Humedad" P21 }
  { "Producto Vectorial" P20 }
  { "Suma de Serie" P12 }
  { "Graficar Ecuacion" P18 }
  { "Parametros hidraulicos" P27 }
  { "Tipo de Objetos" P24 }
  { "Cono Circular" P26 } }
  1 CHOOSE DROP EVAL
»
```

Ejecutamos el programa:

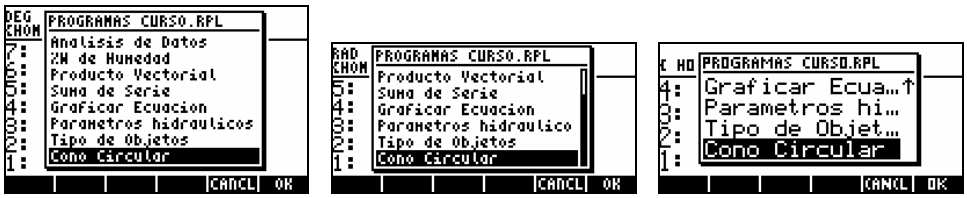
49G+/50G **49G** **48G**



1 CHOOSE *posición inicial por defecto resalta el primer ítem.*

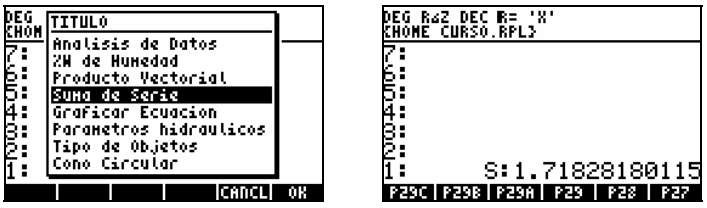


4 CHOOSE *cuarta posición, resalta el cuarto ítem.*



8 CHOOSE *octava posición, resalta el octavo ítem.*

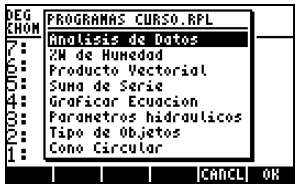
Para ejecutar nuestro programa seleccionamos una opción, cualquiera:

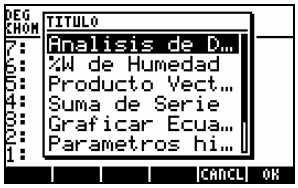


Seleccionamos el cuarto ítem, ejecutara el programa P12

49G/G+/50G

Si el flag 90 esta activado, se muestra el texto con fuente pequeña. Si no esta activado se muestra con una fuente grande como en la HP48





Este flag podemos activarlo, como no, depende de cada usuario y la conveniencia del programador. Para activarlo `-90 SF`

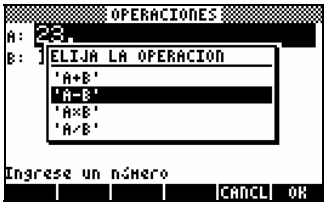
Ejemplo 58: *Programa que calcula una operación seleccionada.*

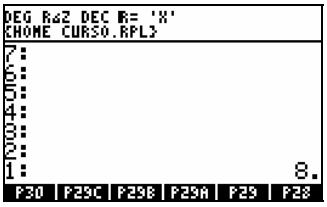
```
P30

« "OPERACIONES"
  { { "A:" "Ingrese un número" 0 }
  { "B:" "Ingrese un número" 0 } }
  { 1 2 } { 2 3 } { }
  INFORM DROP OBJ→ DROP → A B
  « "ELIJA LA OPERACION"
    { { 'A+B' }
    { 'A-B' }
    { 'A*B' }
    { 'A/B' } }
    1 CHOOSE DROP
    OBJ→ DROP EVAL
  »
»
```

Ejecutamos el programa: [F2]

Ingresamos los valores: [2] [3] [ENTER] [1] [5] [ENTER] [ENTER]

Seleccionamos la operación: ENTER





Nótese en el ejemplo, que, estamos insertando en la lista de definiciones un objeto algebraico que se representa en el cuadro, y también es el resultado de la opción seleccionada.

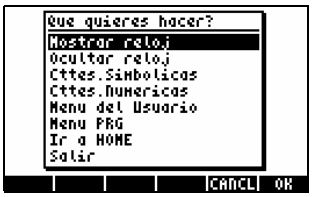
Ejemplo 59: *Ejemplo para mostrar que también se pueden ejecutar (objetos) programas contenidos en la lista de definiciones de CHOOSE.* P31

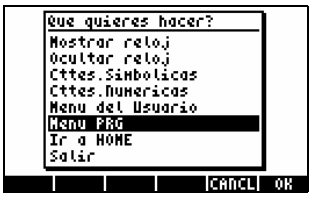
```
« CLLCD
  "Que quieres hacer?"
  { { "Mostrar reloj" « -40 SF » }
  { "Ocultar reloj" « -40 CF » }
  { "Cttes.Simbolicas" « -2 CF » }
  { "Cttes.Numericas" « -2 SF » }
  { "Menu del Usuario" « 2 MENU  » }
  { "Menu PRG" « 22 MENU  » }
  { "Ir a HOME" « HOME » }
  { "Salir" « KILL » } }
  1 CHOOSE DROP EVAL
»
```

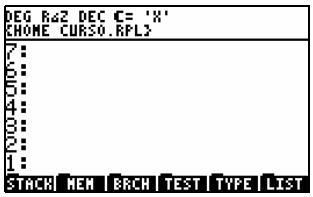
Comandos utilizados:

CLLCD	*Borra la pantalla de la pila,(pero no borra la pila en sí)*
HOME	*Convierte a HOME en el directorio actual que trabajamos*
KILL	*Cancela cualquier programa suspendido*

En el ejemplo se ha insertado pequeñísimos programas en la lista de definiciones, que al ser seleccionadas se ejecutan con EVAL. *Ejecutamos el programa:* ▦F1▦

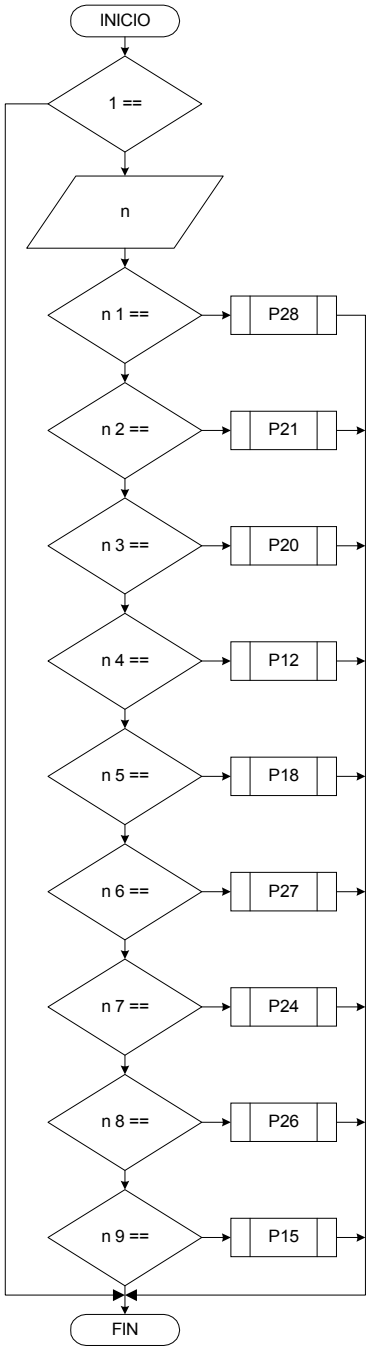




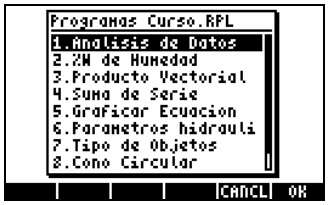


Ejemplo 60: *Programa que muestra, como se puede enumerar las opciones de selección.*

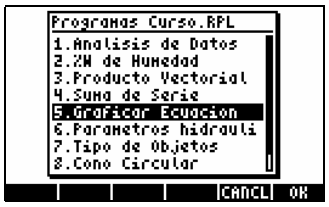
```
P32


« CLLCD
  "Programas Curso.RPL"
  { { "1.Analisis de Datos" 1 }
  { "2.%W de Humedad" 2 }
  { "3.Producto Vectorial" 3 }
  { "4.Suma de Serie" 4 }
  { "5.Graficar Ecuacion" 5 }
  { "6.Parametros hidraulicos" 6 }
  { "7.Tipo de Objetos" 7 }
  { "8.Cono Circular" 8 }
  { "9.PG.Rectangulo" 9 } }
  1
  IF CHOOSE
  THEN → n
    «
      CASE n 1 ==
        THEN P28
        END n 2 ==
        THEN P21
        END n 3 ==
        THEN P20
        END n 4 ==
        THEN P12
        END n 5 ==
        THEN P18
        END n 6 ==
        THEN P27
        END n 7 ==
        THEN P24
        END n 8 ==
        THEN P26
        END n 9 ==
        THEN P15
        END
      END
    »
  ELSE KILL
  END
»
```



Ejecutamos el programa: [EVAL]



Podemos desplazarnos en el cuadro, solo pulsando los números de las opciones, y nos llevara a la opción numerada:

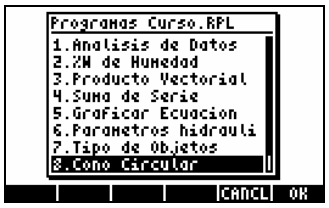


Pulsamos: [5]

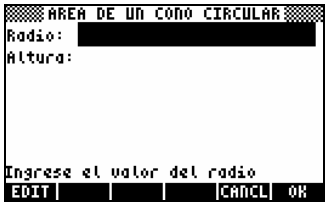
Se remarca la opción cinco

Pulsamos: [8]

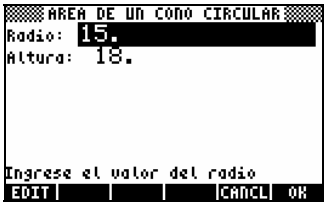
Se remarca la opción ocho

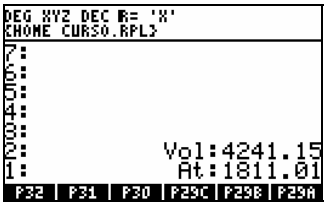


Seleccionamos la opción ocho y presionamos [ENTER]



La opción ocho ejecuta el programa P26, e ingresamos datos para terminar:





15.4. TMENU

Este comando permite crear un menú temporal, el cual podemos utilizar para la entrada de datos. Para crear un menú temporal se sigue la siguiente sintaxis:

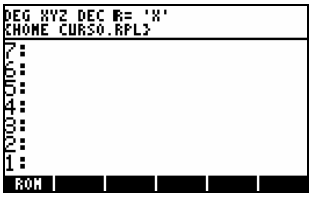
```
« { { "Cadena1" Obj1 } ... { "CadenaN" ObjN } } TMENU »
```

Donde:

"CadenaN"	Nombre que se muestra en un menú
ObjN	Objeto que será devuelto o ejecutado
TMENU	Muestra en pantalla el menú personalizado de listas definidas pero no cambia el contenido de CST.

Ejemplo 61: *Programa que crea un menú temporal para mostrar la versión de la ROM de nuestra calculadora.*

```
P33

« { { "ROM" « VERSION 2 MENU » } } TMENU
»
```

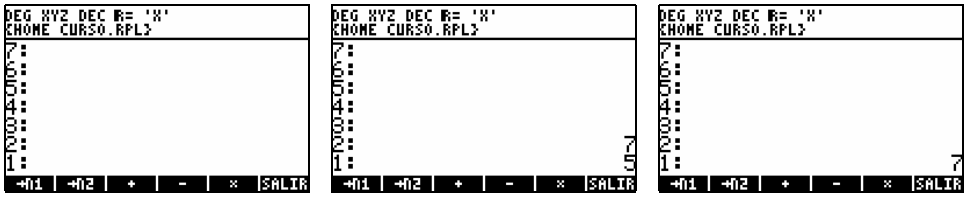


Ejemplo 62: *Programa que asigna y guarda dos objetos, y con otro menú asignado opera los mismos.*

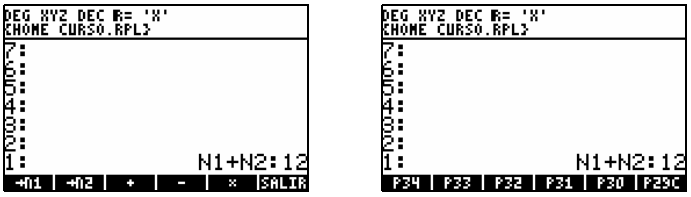
```
P34

« { { "→N1" « 'A' STO » }
  { "→N2" « 'B' STO » }
  { "+" « A B + "N1+N2" →TAG » }
  { "−" « A B − "N1−N2" →TAG » }
  { "*" « A B * "N1*N2" →TAG » }
  { "SALIR" « { A B } PURGE 2 MENU » } }
  TMENU
»
```

Guardamos el programa: '`P34`' STO

Ejecutamos el programa: VAR [F1]
Ingresamos valores: 7 SPC 5 ENTER
Asignamos los valores con los menús: [F1] [F2]



Para operar basta con presionar la operación del menú: [+]
Para terminar el programa: [SALIR]



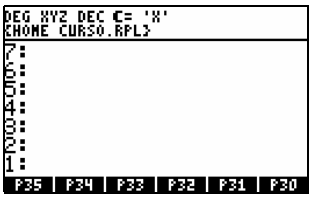
Ejemplo 63: *Programa que incrusta gráficos como menús, podemos mostrarlos con TMENU*

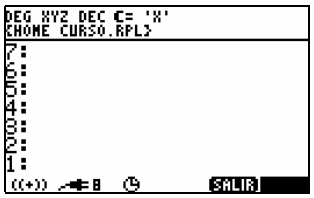
`P35`

```
« {
  { GROB 21 8 0000000000000902108444208 4E4208444200902100000000
  { « -56 SF » « -56 CF » } }
  { GROB 21 8 0000000000000060700CF3500F7070CCF350206070000000
  « SERVER » }
  { GROB 21 8 00000000E000005100084200 08C3000802000011000 0E000
  { « -40 SF » « -40 CF » « TIME 2 RND » } }
  { GROB 21 8 0000000000000000000000000000000000000000000000000000
  }
  { GROB 21 8 000000EFFFF076BBC1BBABA173ABC1FAABA13B2AA1EFFFF0
  « 2 MENU » } { } }
  TMENU
»
```

Guardamos el programa: '`P35`' STO

Ejecutamos el programa: VAR





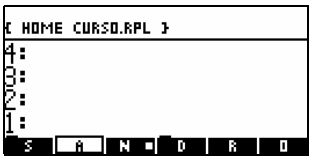
Ejecutamos una opción del menú: F3

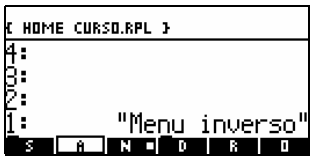
Para terminar el programa: ON

Ejemplo 64: *Programa que crea gráficos como menús (SYSEVAL, solo HP48)*

```
P36

« "S" # 238572d SYSEVAL « "Directorio" » 2 →LIST
"A" # 238670d SYSEVAL « "Menu inverso" » 2 →LIST
"N" # 238474d SYSEVAL « "Estado encendido" » 2 →LIST
"D" # 238572d SYSEVAL « "Directorio" » 2 →LIST
"R" # 238376d SYSEVAL « "Menu normal" » 2 →LIST
"O" # 238376d SYSEVAL « "Menu normal" » 2 →LIST
6 →LIST TMENU
»
```

Ejecutamos el programa: VAR





En este programa usamos menús con códigos con SYSEVAL

A partir de las calculadoras de series 48G se implemento un nuevo comando de nombre SYSEVAL, este comando es igual de peligroso que el comando LIBEVAL, así que les recomiendo usarlo con mucha precaución, el uso indebido de este comando puede ocasionar la perdida del contenido de la memoria de la calculadora, estos vamoa a estudiarlos en el curso de Sys-RPL. SYSEVALs son direcciones que apuntan directamente a un lugar de la ROM de la HP48. Muchos strings hexadecimales SYSEVAL son comandos System-RPL. Ya que no hay revisión de argumentos, es muy fácil borrar la memoria de tu HP.

15.5. PROMPT

Este comando interrumpe la ejecución de un programa y muestra un mensaje en la parte superior de la pantalla de la calculadora. El usuario debe presionar CONT para continuar la ejecución del programa.

```
« "Cadena" PROMPT »
```

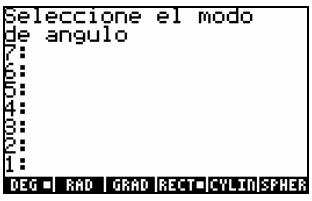
Donde:

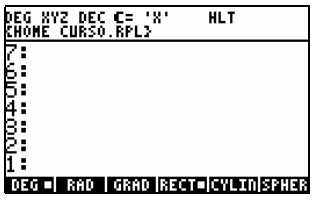
"Cadena"	Mensaje a mostrarse en el área de estado
PROMPT	Muestra la cadena e interrumpe la ejecución del programa.

Ejemplo 65: *Programa que muestra el uso de PROMPT.*

```
P37

« 65 MENU "Seleccione el modo
de angulo" PROMPT
»
```

Ejecutamos el programa: [EVAL]
Seleccionamos el modo de ángulo que queremos trabajar: [DEG]
Para continuar (terminar) el programa presionamos CONT



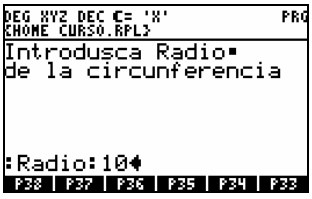


Nótese que en el área de indicadores aparece el indicador HLT *Este comando detiene el programa para continuar el programa hacemos uso del comando* CONT *que termina el programa.*

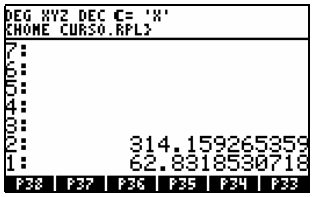
Ejemplo 66: *Programa que hace una combinación de INPUT, TMENU y PROMPT*

```
P38

« "Introdusca Radio
de la circunferencia"
{ ":Radio:" { 1 9 } V } INPUT
OBJ→ DTAG → R
«
{ { "Long" « 2 π R * * →NUM » }
{ "Area" « R 2 ^ π * * →NUM » }
{ } { } { "NUEVO" « 0 MENU P038 CONT » }
{ "SALIR" « 2 MENU CONT » } } TMENU
"Elija del Menú" PROMPT
»
```

Ejecutamos el programa: VAR P38
Ingresamos el valor del radio: 1 0 ENTER



Elegimos una opción del menú: NEXT
Para terminar el programa: ATTN



Ejemplo 67: *Programa que hace una combinación de INPUT, TMENU y PROMPT*

```
P39

« CLLCD "MATRIZ"
{ {GENERAR «"TAMAÑO DE LA MATRIZ"
{ { "Filas:" "Numero de filas" 0 }
{ "Columnas:" "Numero de columnas" 0 } }
{ 1 12 } { 3 3 } { } INFORM DROP RANM »}
{ EDITOR « #A2012h LIBEVAL » } }
1 CHOOSE DROP EVAL
{ { "DET" « DET » }
{ "INV" « INV » }
{ "TRACE" « TRACE »  }
{ } { "NUEVO" « 2 MENU P39 CONT » }
{ "SALIR" « 2 MENU CONT » } } TMENU
"Elija opción del Menú" PROMPT
»
```

 Editor de Matrices

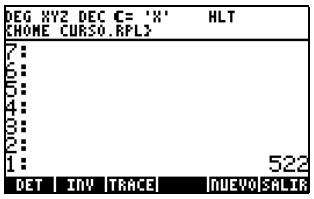
```
« #44C31h SYSEVAL »
```

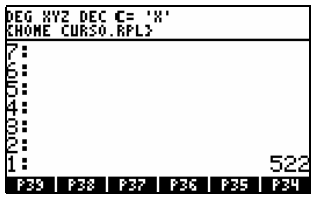
 Editor de Matrices

```
« #A2012h LIBEVAL »
```

Ejecutamos el programa: [F3]
Generamos la matriz: [F1] 3 [SPC] 3 [ENTER] [ENTER]

Elegimos una opción del menú: [F1]
Para terminar el programa: [F6]





16. Salida de datos

Un programa siempre tiene que ser amigable (interface agradable al usuario), cuando elaboramos un programa puede que nos interese presentar de la mejor manera lo datos de salida para informar los resultados de alguna operación o proceso. Para realizar esto la calculadora dispone de comandos destinados para tal fin, y su uso depende de cómo se desee presentar los datos.

Para tener acceso a los comandos de salida de datos presionamos: ◁ PRG NXT ▦ También podemos ir a este menú escribiendo en la pila: `40 MENU`

16.1. MSGBOX

El comando MSGBOX permite crear cuadros de diálogos. Esto recuadros solo tienen como objeto el informar algún mensaje, por lo que solo se cierran oprimiendo la tecla de menú ▦ Para crear una caja de dialogo solo se tiene que seguir esta sintaxis:

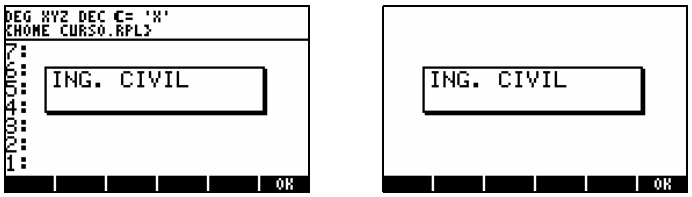
```
« "Cadena" MSGBOX »
```

Donde:

`"Cadena"`	Texto a mostrarse en el cuadro
`MSGBOX`	Crea un recuadro de mensaje definido a partir de una secuencia de caracteres de texto.

Ejemplo 68: *Programa que muestra el uso del comando MSGBOX*

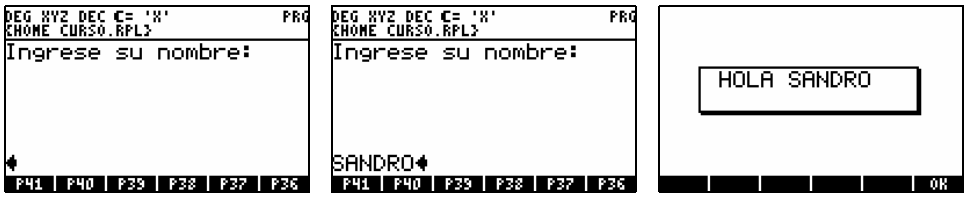
```
P40

« CLLCD "ING. CIVIL" MSGBOX
»
```



La imagen de la derecha muestra el uso del comando `CLLCD` que borra el fondo del display, pero no borra la pila en sí.

Ejemplo 69: *Programa que recoge un objeto con input y adiciona a otro para mostrar el mensaje con MSGBOX*

```
P41

«
 "Ingrse su nombre:"
  { "" α }
  INPUT CLLCD " HOLA " SWAP + MSGBOX
»
```



Ejemplo 70: *Programa que muestra el uso de comandos de TIME*

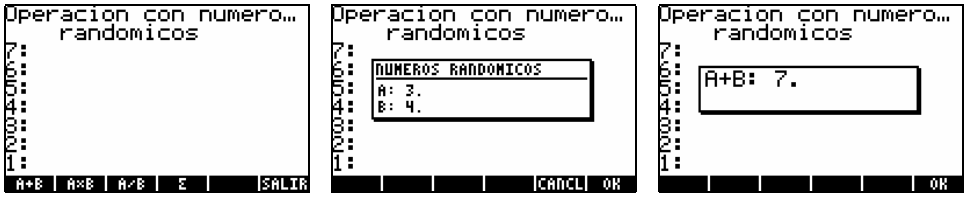
```
P42

« CLEAR
"Dia de Nacimiento?" "" INPUT OBJ→
"Mes?" "" INPUT OBJ→
"Año? (Ej: 1985)" "" INPUT OBJ→ → d m a
« "Tienes " d m 100 / + a 1000000 / + DUP DATE DDAYS SWAP
TIME TSTR 1 3 SUB 'd' STO +
" Dias de Vida " + CLLCD MSGBOX
»
»
```

Para este ejemplo tiene que estar activado el indicador 42 *para el formato de fecha* dd.mm.yy format

Ejemplo 71: *Programa que trabaja con números randomicos (aleatorios)*

```
P43

« RAND 10 * IP RAND 10 * IP → A B
  « { { "A+B" « "NUMEROS RANDOMICOS"
    "A: " A →STR + 1 →LIST
    "B: " B →STR + 1 →LIST 2 →LIST
    0 CHOOSE DROP "A+B: " A B + →STR + MSGBOX » }
    @AB*
    { "A*B" « "NUMEROS RANDOMICOS"
    "A: " A →STR + 1 →LIST
    "B: " B →STR + 1 →LIST 2 →LIST
    0 CHOOSE DROP "A*B: " A B * →STR + MSGBOX » }
    @AB/
    { "A/B" « "NUMEROS RANDOMICOS"
    "A: " A →STR + 1 →LIST
    "B: " B →STR + 1 →LIST 2 →LIST
    0 CHOOSE DROP "A/B: " A B / →STR + MSGBOX » }
    @ABΣum
    { "Σ" « "NUMEROS RANDOMICOS" "A: " A →STR + 1 →LIST
    "B: " B →STR + 1 →LIST 2 →LIST
    0 CHOOSE DROP "S=a+..+b: "
    'Σ(X=A,B,X)' EVAL →STR + MSGBOX » } {  }
    { "SALIR" « 2 MENU CONT » }
    } TMENU
    "Operacion con numeros
    randomicos" PROMPT
  »
»
```

En este programa se muestra como se convierte un objeto real a una cadena con el comando →STR



Comandos utilizados:

RAND	*Devuelve un número aleatorio*
IP	*Parte entera de un número real*
→STR	*Convierte un objeto en una cadena*
→LIST	*Convierte los objeto entre el nivel 1 y el nivel actual en una lista de N Objetos*

16.2.…DIPS…WAIT

Este comando muestra un objeto en una línea de la pantalla el número de línea puede ser de 1 a 7 para la HP48 y de 1 a 9 para la HP49/50. Para este proceso necesitamos dos argumentos primero una cadena y luego el número de línea. La sintaxis a seguir es

```
« "Cadena" #linea DISP #T WAIT »
```

Donde:

"Cadena"	Texto a mostrarse en una línea de la pantalla
#linea	Número de fila (línea) donde se muestra un objeto.
DISP	Comando que muestra en pantalla un objeto
#T	Número expresado en segundos
WAIT	Interrumpe la ejecución la ejecución de un programa durante un número especificado de segundos o hasta que se pulse una tecla.

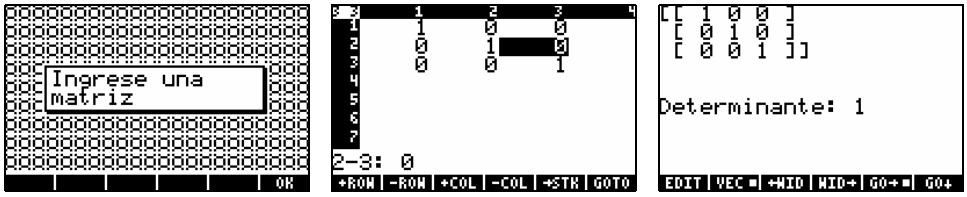
Ejemplo 72: *Programa que muestra en la pantalla una cadena por tres segundos.*

```
P44

« "INGENIERIA CIVIL"
  2 DISP 3 WAIT
»
```


```
P44a

« "INGENIERIA ELECTRICA"
{ 1 3 } DISP 3 WAIT
»
```

Ejemplo 73: *Programa que muestra en pantalla una matriz*

```
P45

« @ #44C31h SYSEVAL HP48
  @ #A2012h LIBEVAL HP49
  "ααααααααααααααααααααααα"
  ⟨ 1 2 3 4 5 6 7⟩ DISP
  "Ingrese una matriz" MSGBOX
  #A2012h LIBEVAL DUP
  DET "Determinante" →TAG
  2 →LIST
  CLLCD ⟨ 1 4 ⟩ DISP 0 WAIT DROP
»
```

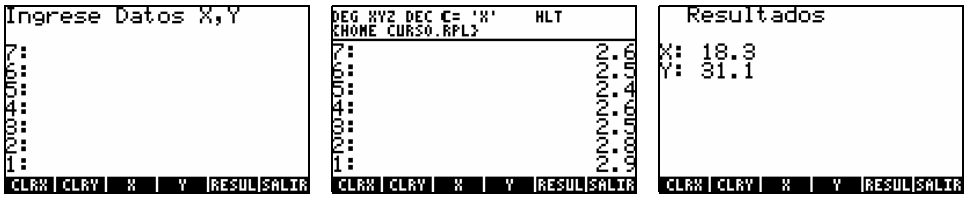


Ejemplo 74: *Programa que realiza la suma de coordenadas X, Y, presenta objetos etiquetados en la pantalla.*

```
P46

« 0 'X' STO 0 'Y' STO
  ⟨
  ⟨ "CLRX" « 0 'X' STO » ⟩
  ⟨ "CLRY" « 0 'Y' STO » ⟩
  ⟨ "X" « 'X' STO+ » ⟩
  ⟨ "Y" « 'Y' STO+ » ⟩
  ⟨ "RESULTADOS" « CLLCD
    "  Resultados" 1 DISP
    X "X" →TAG Y "Y" →TAG
    2 →LIST
    ⟨ 3 4 ⟩ DISP
    0 WAIT DROP
    ⟨ X Y ⟩ PURGE » ⟩
  ⟨ "SALIR" « 2 MENU
  ⟨ X Y ⟩ PURGE CONT » ⟩
  ⟩ TMENU
  "Ingrese Datos X,Y"
  PROMPT
»
```

X	Menú	Y	Menú
2.6	▓	4.0	▓
2.5	▓	4.6	▓
2.4	▓	4.3	▓
2.6	▓	4.2	▓
2.5	▓	4.8	▓
2.8	▓	4.5	▓
2.9	▓	4.7	▓
Σ:18.3		Σ:31.1	

El comando STO+ suma un número u otro objeto al contenido de una variable especificada para nuestro caso X, Y



16.3.…DISP…FREEZE

FREEZE se encarga de congelar la pantalla en un área determinada y que se le proporciona como argumento un número. La sintaxis a seguir es:

```
« "Cadena" #linea DISP #parte FREEZE »
```

Donde:

"Cadena"	Texto a mostrarse en una línea de la pantalla
#linea	Número de fila (línea) donde se muestra un objeto.
DISP	Comando que muestra en pantalla un objeto
#parte	Número que indica un área especifica de la pantalla
FREEZE	Comando que congela un área de la pantalla hasta que se pulsa una tecla.

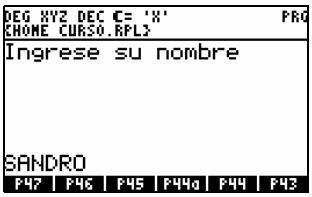
Los números que determinan el área de la pantalla son:

Número	Zona Congelada
0	Toda la pantalla
1	Área de Estado
2	La pila
3	Área de Estado y la Pila
4	Los menús
5	Área de Estado y los menús
6	La pila y los menús
7	Toda la pantalla

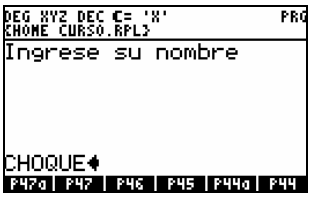
Los números mayores de 7 o menores de 0 son equivalentes a 7

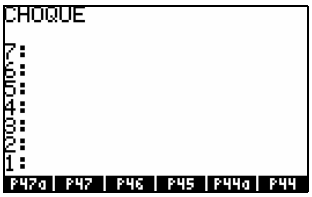
Ejemplo 75: *Programa que muestra en uso de FREEZE*

```
P47

« "Ingrese su nombre"
  { "" { 0 0 } α }
  INPUT
  1 DISP 1 FREEZE
»
```



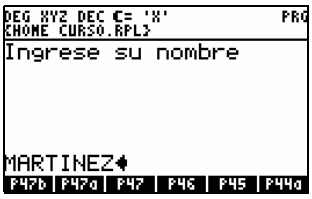

```
P47a

« "Ingrese su nombre"
  { "" { 0 0 } α }
  INPUT
  CLLCD 1 DISP 1 FREEZE
»
```

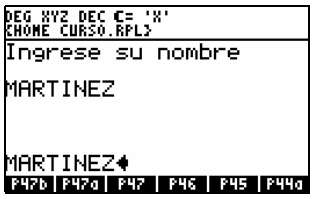




Nótese que en el programa `P47a` el área de estado no se borra, pero en el programa `P47a` el área de estado se borra, esto lo hicimos con el comando `CLLCD` Normalmente así a la vista no se nota la congelación de la pantalla pero con estos dos ejemplos hemos aclarado todo.

```
P47b

« "Ingrese su nombre"
  { "" { 0 0 } α }
  INPUT → N
  « N 5 DISP 2 FREEZE »
»
```

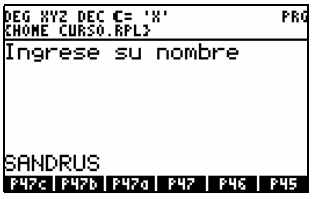


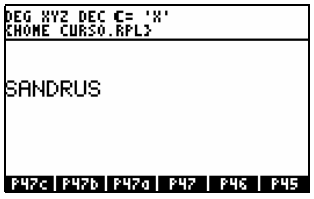


En el programa P47b se congela con FREEZE el área de la pila (la pila no se borra)

P47c

```
« "Ingrese su nombre"
  { "" { 0 0 } α  }
  INPUT → N
  « CLLCD N 5 DISP 2 FREEZE »
»
```





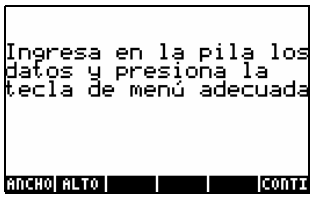
En el programa P47c se congela con FREEZE el área de la pila (la pila si se a borrado)

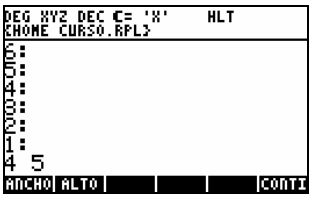
Ejemplo 76: *Programa que muestra en uso de FREEZE con HALT*

P48

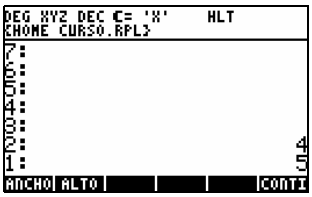
```
« RCLMENU → menu
  «
   { { "ANCHO" « 'AN' STO » } { "ALTO" « 'AL' STO » }
   { } { } { } { "CONTINUAR" « CONT » } } TMENU CLLCD
"Ingresa en la pila los
datos y presiona la
tecla de menú adecuada"
3 DISP 3 FREEZE HALT
AN "Ancho" →TAG
AL "Alto" →TAG
AL AN * "Area" →TAG 3 →LIST
CLLCD { 2 3 5 } DISP 0 WAIT DROP
{ AL AN } PURGE
menu MENU
  »
»
```

Guardamos el programa: '`P48`' STO▶
Ejecutamos el programa: VAR ▓▓:▓
Ingresamos valores: 7 SPC 5 ENTER
Asignamos los valores con los menús: ▓▓▓ ▓▓▓





Para ver resultados: ▓▓▓
Para terminar el programa: ▓▓▓



16.4. PVIEW

Este comando cambia a la pantalla en modo gráfico, con coordenadas de puntos especificados, solo se puede salir con ON (CANCEL

La sintaxis a seguir es:

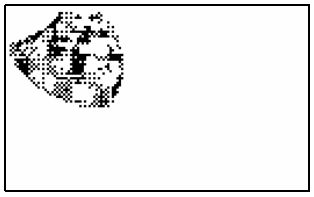
```
« Objetograf { Coordenadas 0 0 } PVIEW »
```

Donde:

ObjetoGraf	Objeto grafico a mostrarse en la pantalla
Coordenadas 0 0	Coordenadas que especifican la esquina superior izquiera { # 0d # 0d }
PVIEW	Comando que muestra PICT en la pantalla

Ejemplo 77: *Programa que muestra en uso de PVIEW y con WAIT interrumpimos el programa para ver el gráfico.*

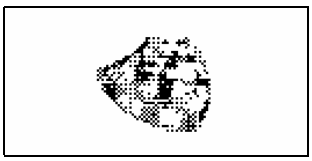
```
P49

« Graphic 57 x 49
  PICT STO ( # 0d # 0d ) PVIEW 0 WAIT DROP
»
```

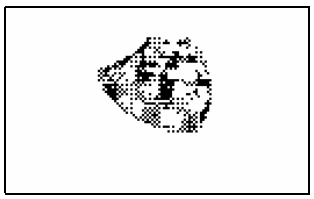


Ejemplo 78: *Programa que muestra en uso de PVIEW.*

```
P50
```

Código fuente	Comentarios
«	*Inicio del programa*
PICT	*Coloca PICT en la pila*
PURGE	*Limpia PICT*
PICT	*Coloca PICT en la pila*
(# 38d # 10d)	*Nuevas coordenadas del gráfico*
Graphic 57 x 49	*Objeto gráfico*
GOR	*Comando que superpone el objeto gráfico sobre otro con las nuevas coordenadas*
( )	*Coordenadas de la parte izquierda superior*
PVIEW	*Comando que muestra PICT*
0 WAIT	*Se interrumpe el programa hasta presionar una tecla.*
DROP	*Borra el objeto del nivel -1*
»	*Fin del programa*

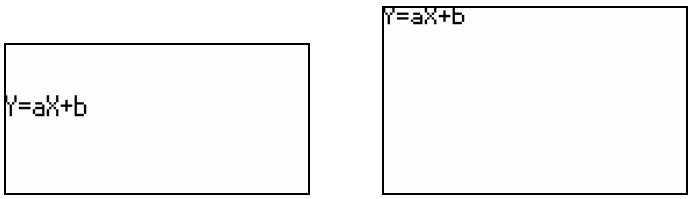




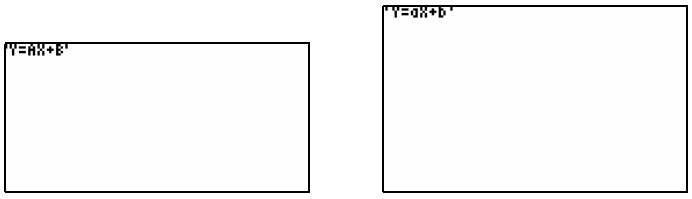
No siempre se presentan gráficos, a continuación se muestra la conversión de un objeto algebraico, cadena, listas y programas en un objeto gráfico.

Ejemplo 79: *Programa que muestra la conversión de un objeto (x) a un objeto gráfico con el comando →GROB*

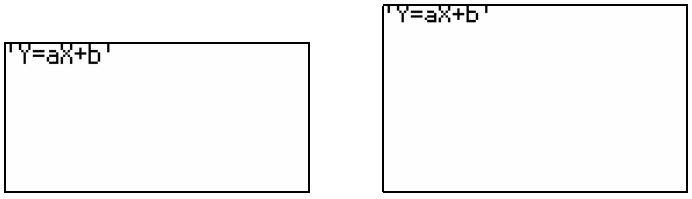
```
P51

« « 'Y=aX+b' 0 →GROB PICT STO
  ( # 0d # 0d ) PVIEW 0 WAIT DROP
»
```




```
P51a

« « 'Y=aX+b' 1 →GROB PICT STO
  ( # 0d # 0d ) PVIEW 0 WAIT DROP
»
```




```
P51b

« « 'Y=aX+b' 3 →GROB PICT STO
  ( ) PVIEW 0 WAIT DROP
»
```



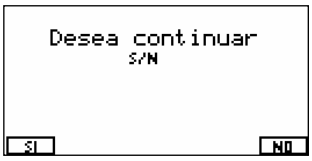
Cuando se este colocando ( ) para el comando PVIEW se sobreentiende que es la coordenada izquierda superior equivale

El comando →GROB
Con este comando puedes convertir en gráficos objetos como cadenas de texto, nombres, números, listas programas, etc.
El gráfico puede ser creado como grandes y medianos.
El comando →GROB requiere un objeto y un número el que indica el tamaño de caracteres.

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

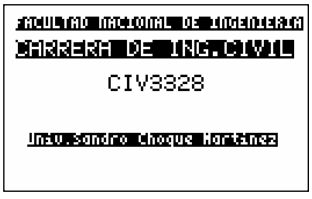
Ejemplo 80: *Programa que muestra en pantalla para tomar una decisión y darle alguna dirección después de presionar una tecla.*

```
P52

« ERASE
  PICT ( # 20d # 10d )
  "Desea continuar" 2 →GROB GOR
  PICT ( # 54d # 20d )
  "S/N" 1 →GROB GOR
  PICT ( # 1d # 56d )
  "SI" #238670d SYSEVAL GOR
  PICT ( # 110d # 56d )
  "NO" #238670d SYSEVAL GOR
  ( ) PVIEW 0 WAIT DROP
»
```



Ejemplo 81: *Programa que muestra los tipos de letras y manejo de coordenadas.*

```
P53

« ERASE PICT ( # 5d # 5d )
  "FACULTAD NACIONAL DE INGENIERIA"
  1 →GROB NEG GOR PICT ( # 5d # 15d )
  "CARRERA DE ING.CIVIL"
  2 →GROB NEG GOR PICT
  ( # 45d # 30d )
  "CIV3328" 2 →GROB GOR
  PICT ( # 10d # 55d )
  "Univ.Sandro Choque Martinez" 1 →GROB NEG GOR
  ( ) PVIEW 0 WAIT DROP
»
```



16.5. BEEP

Comando que emite un sonido mediante una frecuencia y un determinado tiempo. Solo funciona si esta activado el indicador esta activado.
Su sintaxis es:

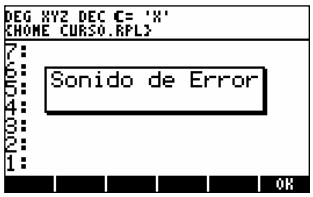
```
« Frecuencia Tiempo BEEP »
```

Donde:

Frecuencia	Frecuencia dada en Hertz.
Tiempo	Tiempo de duración en segundos
BEEP	Comando que emite pitidos a una frecuencia y un tiempo.

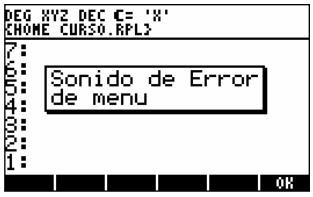
Ejemplo 82: *Sonido de error del sistema.*

```
P54

« 1392 0.075 BEEP
"Sonido de Error" MSGBOX »
```



Ejemplo 83: *Sonido de error del menú.*

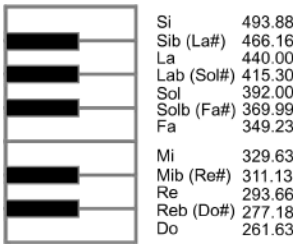
```
P55

« 337 0.07 BEEP
"Sonido de Error de menu" MSGBOX »
```



Ejemplo 84: *Sonido de un reloj cucu*

```
P56

« @ relojcucu
  1200 .1 BEEP
  .2 WAIT 1000 .1 BEEP
  1 WAIT 1200 .1 BEEP
  .2 WAIT 1000 .1 BEEP
»
```

Podemos componer excelentes melodías, con el uso de algunos comandos más; como en el ejemplo 84, que insertamos con WAIT un silencio y la frecuencia de cada nota. Para saber las notas musicales, solo debes conocer la frecuencia de cada nota y eso hay en cualquier sitio de música o manual de un instrumento, pero aquí los tienes, haber si creas alguna melodía.



Ejemplo 84: *Pitidos del tema de la pantera rosa.*

```
P57

« 165 .25 BEEP .25 WAIT 185 .125 BEEP
196 .25 BEEP .25 WAIT 156 .125 BEEP
165 .125 BEEP .125 WAIT 185 .125 BEEP
196 .125 BEEP .125 WAIT 262 .125 BEEP
247 .125 BEEP .125 WAIT 165 .125 BEEP
196 .125 BEEP .125 WAIT 247 .125 BEEP
233 .75 BEEP 220 .125 BEEP 196 .125
BEEP 165 .125 BEEP 147 .125 BEEP 165
.5 BEEP .5 WAIT 156 .125 BEEP 165 .25
BEEP .25 WAIT 185 .125 BEEP 196 .25
BEEP .25 WAIT 156 .125 BEEP 165 .125
BEEP .125 WAIT 185 .125 BEEP 196 .125
BEEP .125 WAIT 262 .125 BEEP 247 .125
BEEP .125 WAIT 196 .125 BEEP 247 .125
BEEP .125 WAIT 330 .125 BEEP 311 .75
BEEP .5 WAIT 156 .125 BEEP 165 .25
BEEP .25 WAIT 185 .125 BEEP 196 .25
BEEP .25 WAIT 156 .125 BEEP 165 .125
BEEP .125 WAIT 185 .125 BEEP 196 .125
BEEP .125 WAIT 262 .125 BEEP 247 .125
BEEP .125 WAIT 165 .125 BEEP 196 .125
BEEP .125 WAIT 247 .125 BEEP 233 .75
BEEP 220 .125 BEEP 196 .125 BEEP 165
.125 BEEP 147 .125 BEEP 165 .5 BEEP .5
WAIT 330 .125 BEEP .125 WAIT 294 .125
BEEP 247 .125 BEEP .125 WAIT 220 .125
BEEP 196 .125 BEEP .125 WAIT 165 .125
BEEP 233 .125 BEEP 220 .125 BEEP .125
WAIT 233 .125 BEEP 220 .125 BEEP .125
WAIT 233 .125 BEEP 220 .125 BEEP .125
WAIT 233 .125 BEEP 220 .125 BEEP .125
WAIT 196 .125 BEEP 165 .125 BEEP 147
.125 BEEP 165 .125 BEEP 165 .5 BEEP .75 WAIT
»
```

Capítulo IV

Estructuras de Programación

La programación estructurada se refiere a un tipo de programación que produce código con un flujo limpio, un diseño claro y un cierto grado de modularidad o de estructura jerárquica. Entre los beneficios de la programación estructurada se encuentran la facilidad de mantenimiento y la legibilidad por parte de otros programadores.

La programación estructurada se refiere tanto a la estrategia del programador, como al lenguaje utilizado. Así, para ella se usan lenguajes de programación que faciliten el diseño de aplicaciones con llamadas a procedimientos o funciones, como lo son típicamente Pascal y Ada, entre otros. En ellos el flujo de información es más fácilmente legible y no requieren de bifurcaciones basadas en llamadas a líneas concretas, sino, más bien, en saltos a áreas de código perfectamente diferenciadas. La programación bien estructurada permite, de forma adicional, la reusabilidad del código, extrayendo módulos que pueden ser utilizados en otros programas, sin cambios en el código o con un mínimo de readaptaciones.

Imagina que estamos creando un pequeño programa para un robot que debe subir 10 escalones. El robot entiende las siguientes instrucciones LevantaPieIzquierdo (para levantar el pie izquierdo y subir un escalón) y LevantaPieDerecho (para levantar el pie derecho y subir otro escalón), con lo que podrá ir ascendiendo hasta llegar al final de la escalera. Si solo pudiésemos utilizar estas dos instrucciones deberíamos hacer un programa con las siguientes líneas de código:

```
LevantaPieIzquierdo
LevantaPieDerecho
LevantaPieIzquierdo
LevantaPieDerecho
LevantaPieIzquierdo
LevantaPieDerecho
LevantaPieIzquierdo
LevantaPieDerecho
LevantaPieIzquierdo
LevantaPieDerecho
```

Fíjate que en este caso hemos tenido que escribir las mismas instrucciones varias veces para que el robot fuera subiendo la escalera. ¿Que hubiese sucedido si el robot en lugar de subir 10 escalones hubiese tenido que subir la Torre Eifiel?. Pues que el código hubiese sido interminable, corriendo el peligro de equivocarnos al contar la cantidad de escalones, con lo que el robot no hubiese llegado a la cima de la escalera. O incluso nos podríamos haber equivocado poniendo dos veces la misma instrucción, con lo que el robot se hubiese pegado un golpetazo impresionante al levantar dos veces el mismo pie.

Para solucionar estos problemas disponemos de diferentes instrucciones que nos permiten reducir el número de líneas de un programa facilitando así la compresión, la modificación del código y un posible error en la ejecución del programa.

Observa una solución para nuestro problema. (Piensa que para un mismo problema no solo existe una solución, yo te ofreceré una, pero eso no quiere decir que sea la mejor). Las instrucciones se explicarán con detenimiento posteriormente.

```
Repetir hasta que NúmeroEscalón = 10
   LevantaPieIzquierdo
   LevantaPieDerecho
   Suma 1 a NúmeroEscalón
Fin Repetir
```

17. Estructuras de Selección

En la solución de la mayoría de los problemas algorítmicos se requieren efectuar tomas de decisiones que conducen a la ejecución de una o más acciones dependiendo de la verdad o falsedad de una o más condiciones. Como consecuencia de esto se producen cambios en el flujo de control del programa. Dicho flujo de control implica rutas que deben ser seleccionadas. Para esto, se utilizan ciertas estructuras de programación conocidas como estructuras selectivas o estructuras de decisión.

Las estructuras condicionales permiten que un programa tome una decisión basada en el resultado de una o más pruebas. Los comandos para estas estructuras se encuentran en el menú [PRG] [BRCH] También ingresamos al menú, escribiendo en la pila `24 MENU`

17.1. Estructuras de selección simple

17.1.1. IF…THEN…END

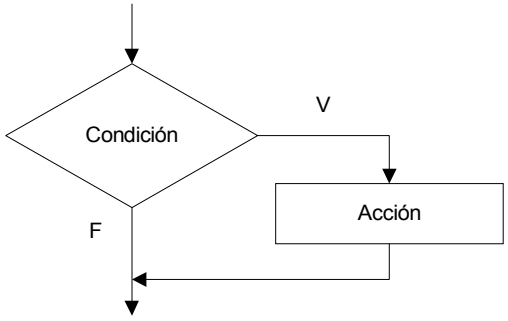
Traducción: **SI...ENTONCES...FIN**

En la estructura de selección simple SI, evalúa una condición lógica

- Si la condición es verdadera se ejecuta la acción. La acción puede ser una acción simple (una sola acción) o una acción compuesta (conjunto de acciones).
- Si la condición es falsa, no se hace nada.

Sintaxis:

```
« ...
  IF Condición THEN
     Acción
  END
  ...
»
```

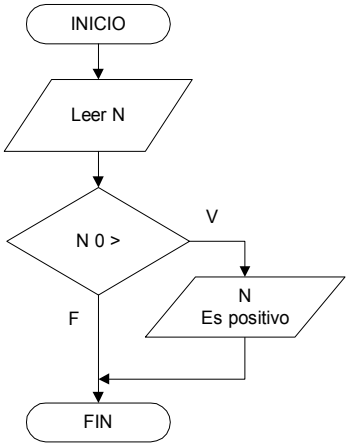


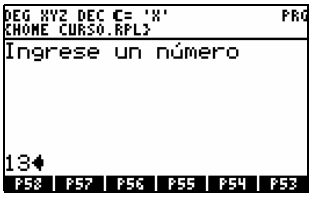
IF…THE…END ejecuta la secuencia de comandos de la *acción* solo si la *condición* es verdadera. La condición puede ser una secuencia de comandos (por ejemplo, `A B ≤`) o una operación algebraica (por ejemplo, `'A≤B'`). Si la condición es una operación algebraica, da automáticamente como un resultado un numérico, no necesita [→NUM] ni [EVAL]

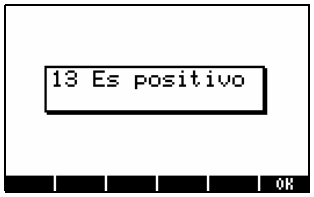
IF inicia la condición, la cual deja un resultado de prueba en la pila. THEN retira el resultado de prueba de la pila. Si se cumple el valor de la condición, se ejecuta la acción; de lo contrario, la ejecución del programa se reanuda después de END.

Ejemplo 85.

```
P58

« "Ingrese un número"
  { "" { 1 0 } V }
  INPUT OBJ→ → N
  « IF N 0 > THEN
    CLLCD N →STR
    " Es positivo" + MSGBOX
    END
  »
»
```



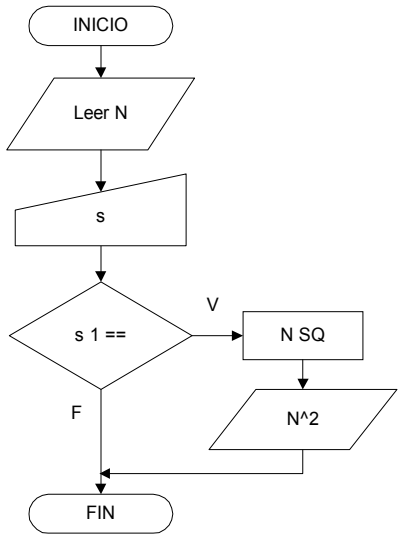




«	*Inicio del programa*
"Ingrese un número" { "" { 1 0 } V } INPUT OBJ→	*Ingreso de N con INPUT*
→ N	*Asigna a N como variable local*
«	*Inicio del subprograma*
IF N 0 >	*IF evalúa la condición lógica (1)*
THEN	*Si se cumple la condición, Entonces:*
CLLCD	*Se Borra la pantalla*
N →STR	*Convierte N a cadena*
" Es positivo" + MSGBOX	*Se adiciona otra cadena y con MSGBOX se muestra en un cuadro de mensaje*
END	*Fin de la estructura IF*
»	*Fin del subprograma*
»	*Fin del programa*

Ejemplo 86. *Programa que eleva al cuadrado un número.*

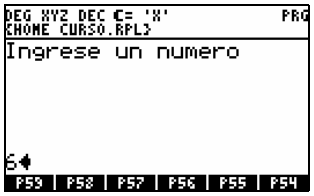
```
P59

« "Ingrese un numero"
  { "" { 1 0 } V }
  INPUT OBJ→ c n
  « CLLCD "Elevar al cuadrado"
    { { SI  1 }
    { NO 2 } } 1
    CHOOSE DROP → s
    « IF s 1 == THEN
      n SQ n →STR "^2" + →TAG
      END
    »
  »
»
```

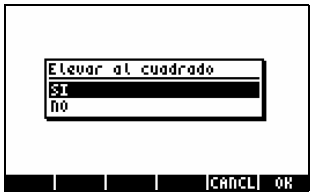


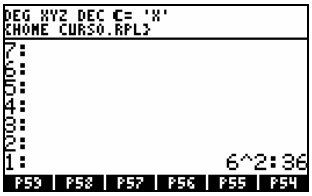
Ejecutamos el programa: ▦▦▦

1: 6 ENTER
2: ▦▦

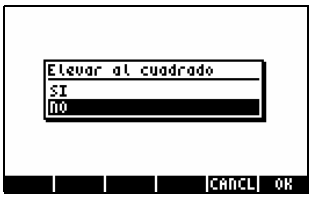
1: 6 ENTER
2: ▦▦

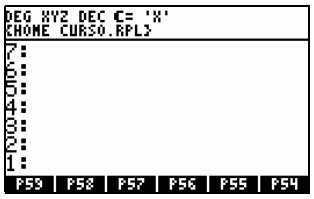






Que pasa si elegimos la opción no: ▼ ENTER





El programa termina no hace ninguna acción, termina el programa.

Las estructuras de selección no siempre pueden realizar una acción, sino más bien pueden realizar una acción compuesta o se a un conjunto de acciones. En el cual puede estar una serie de operaciones un procedimiento que nos llevara a un resultado para nuestra necesidad.

```
IF  Condición  THEN
    Acción A1
    Acción A2
    .
    Acción An
ELSE
    Acción B1
    Acción B2
    .
    Acción Bn
END
```

17.2. Estructuras de selección doble

17.2.1. IF…THEN…ELSE…END

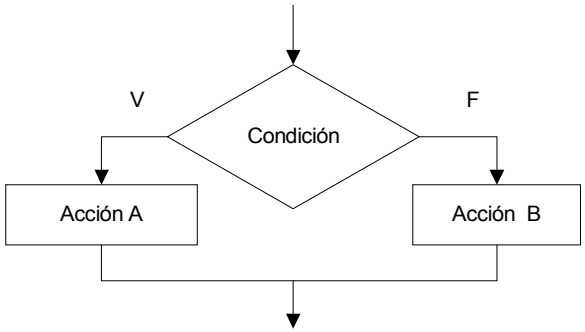
Traducción: **SI...ENTONCES...SINO…FIN**

La estructura de selección doble SI - SINO evalúa una condición lógica:

- Si la condición es verdadera, ejecuta una acción A.
- Si la condición es falsa, ejecuta la acción B.
- Tanto la acción A como la acción B pueden ser acciones simples (una sola acción) o acciones compuestas (un conjunto de acciones).

Sintaxis:

```
« ...
  IF  Condición  THEN
      Acción A
  ELSE
      Acción B
  END
  ...
»
```

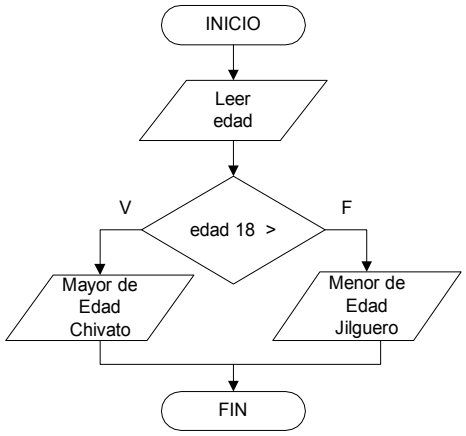


IF…THE…ELSE…END ejecuta la secuencia de comandos de la *acción A* solo si la *condición* es verdadera, o bien la secuencia de comandos de la *acción B* si la *condición* es falsa. Si la condición es una operación algebraica, da como resultado un número no necesita de EVAL ó →NUM

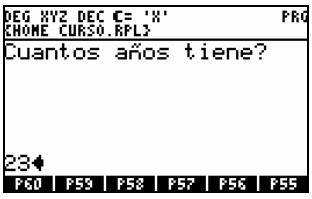
IF inicia la condición, la cual deja un resultado de prueba en la pila, THEN retira el resultado de prueba de la pila. Si el valor es no nulo, se ejecuta la acción A; de lo contrario, se ejecuta la acción B. Una vez ejecutada la acción apropiada, la ejecución se reanuda después de END.

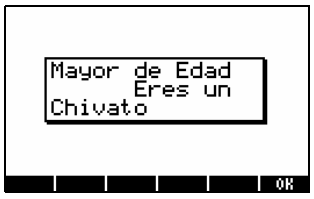
Ejemplo 87. *Programa que determina si una persona es mayor de edad.*

```
P60

« "Cuantos años tiene?"
  { "" { 1 0 } V }
  INPUT OBJ→ → edad
  « IF edad 18 > THEN
      CLLCD "Mayor de Edad
      Eres un Chivato"
      MSGBOX
    ELSE
      CLLCD "Menor de Edad
      Eres un Jilguero"
      MSGBOX
    END
  »
»
```

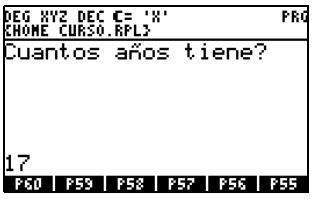


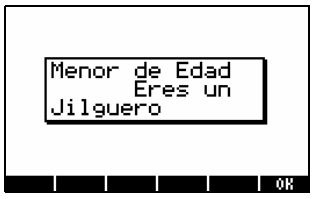
Si ingresamos un número mayor a 18, ejecutara la Acción verdadera.





Si ingresamos un número menor a 18, ejecutara la Acción falsa.

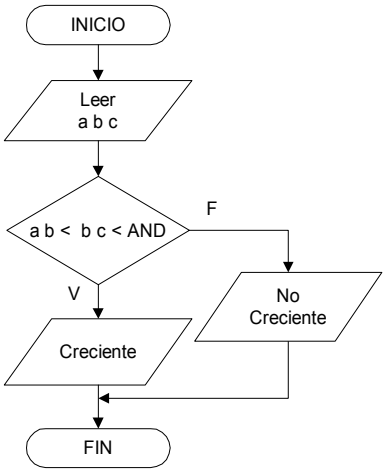




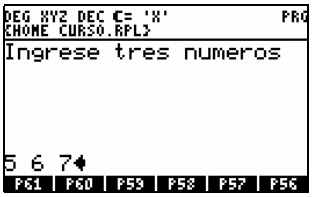
Ejemplo 88. *Programa que detecta si se han ingresado 3 números en forma creciente.*

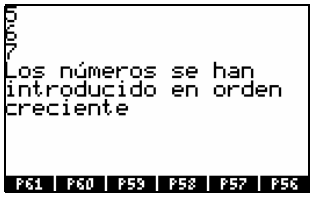
P61

```
« "Ingrese tres numeros"
  { "" { 1 0 } V }
  INPUT OBJ→ → a b c
  « IF 'a<b' 'b<c' AND THEN
      CLLCD
      a b c
      "Los números se han"
      "introducido en orden"
      "creciente"
      6 →LIST { 1 2 3 4 5 6 }
      DISP 0 WAIT DROP
    ELSE
      CLLCD
      a b c
      "Los números NO se han"
      "introducido en orden"
      "creciente" 6 →LIST
      { 1 2 3 4 5 6 }
      DISP 0 WAIT DROP
    END
  »
»
```



En este ejemplo estamos haciendo uso el operador lógico AND *que hace una comparación entre dos argumentos como una condición.* AND *devuelve verdadero si ambos argumentos son verdaderos, se ejecuta la acción verdadera.*

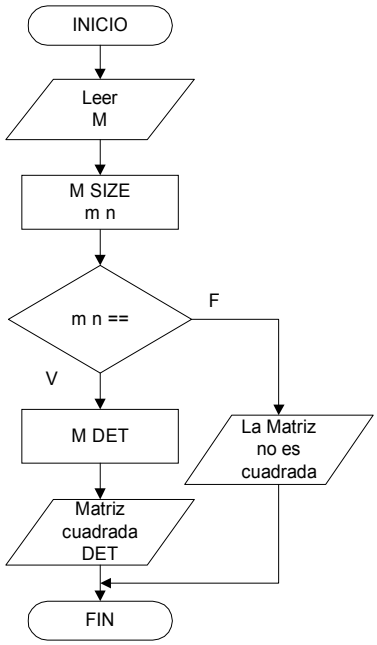




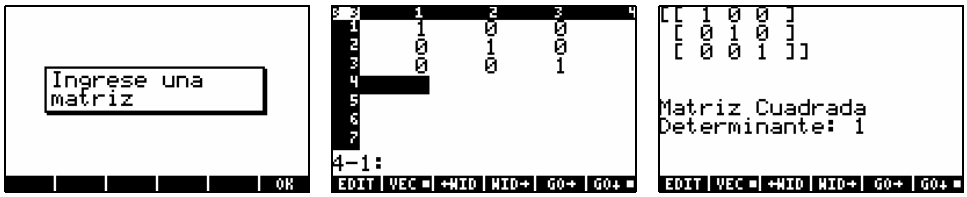
Para expresar cláusulas de prueba, como en el caso de las condiciones de las estructuras selectivas, se requieren de operadores relacionales y operadores lógicos, que se muestran en el capitulo I Págs. 21 y 22 respectivamente.
*Dichas condiciones solo puede tomar uno de los siguientes valores: verdadero (*1*) ó falso (*0*).*

Ejemplo 89. *Programa que encuentra el determinante de una matriz cuadrada*

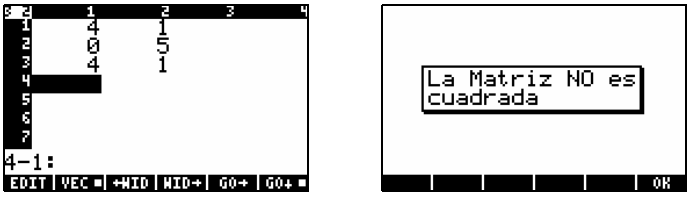
```
P62

« @HP48 #44C31h SYSEVAL
  @HP49 #A2012h LIBEVAL
  CLLCD "Ingrese una matriz"
  MSGBOX #44C31h SYSEVAL
  DUP SIZE
  OBJ→ DROP → M m n
  « IF m n == THEN
      CLLCD
      M "Matriz Cuadrada"
      M DET "Det" →TAG
      3 →LIST { 1 6 7 } DISP
      0 WAIT DROP
   ELSE
      CLLCD
      "La Matriz no es Cuadrada"
      MSGBOX
    END
  »
»
```



Ejecutamos el programa e ingresamos una matriz cuadrada:
Hace una comparación de `m` *x* `n` *si es verdadero (*`1`*) se ejecuta la cláusula verdadera:*



Si no ingresamos una matriz cuadrada se ejecuta otra acción:



Se dice que varias estructuras de selección están en cascada cuando la instrucción que sigue a un IF es otro IF a excepción del último ELSE.

No hay límite en cuanto al número de estructuras de selección doble que pueden ponerse en cascada.

Sintaxis:

```
« ...
  IF Condición1 THEN
     Acción A1
  ELSE
    IF Condición2 THEN
       Acción A2
    ELSE
      IF Condición3 THEN
         Acción A3
      ELSE
        IF CondiciónN THEN
           Acción AN
        ELSE ...
          IF...
            ...
          ELSE
            ...Acción BN
          ...
          .
        END
      END
    END
  END
  ...
»
```

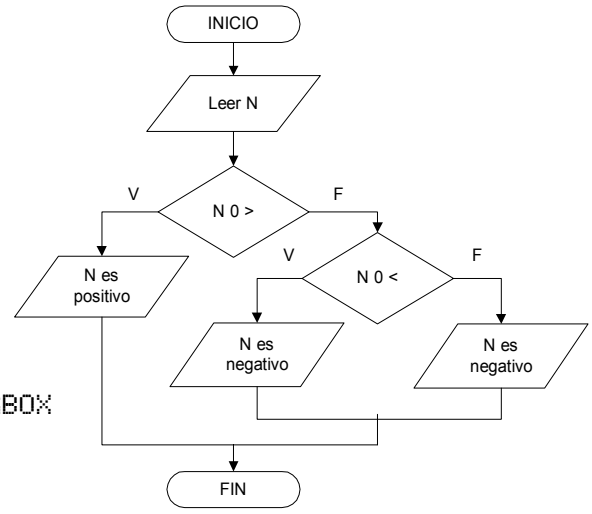
Funcionamiento:

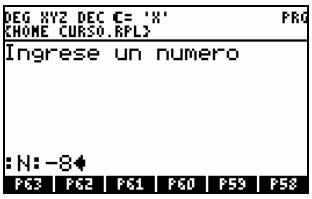
Las condiciones se evalúan en orden descendente pasando de una a otra si la anterior resulta falsa. En el momento que se encuentra una condición verdadera, se efectúa la acción correspondiente a dicha condición y se corta el resto de la estructura.

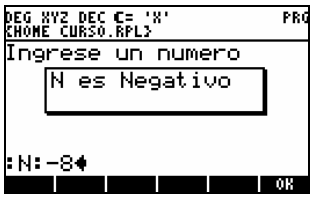
Si todas las condiciones resultan falsas se efectúa la acción correspondiente al último ELSE

Ejemplo 90. *Programa que determina si un número es negativo, positivo o cero*

```
P63

« "Ingrese un numero"
  { ":N:" { 1 4 } V }
  INPUT OBJ→ DTAG → N
  « IF N 0 > THEN
      "N es Positivo"
      MSGBOX
    ELSE
      IF N 0 < THEN
        "N es Negativo"
        MSGBOX
      ELSE
        "N es Cero" MSGBOX
      END
    END

  »
»
```







Ejemplo 91. *Programa que calcula las raíces de la ecuación de segundo grado.*

$$ax^2 + bx + c = 0$$

Para este ejemplo tomamos en cuenta los siguientes casos:

- *Si a es igual a 0 y b es igual a 0, imprimiremos un mensaje diciendo que la ecuación es degenerada.*
- *Si a es igual a 0 y b no es igual a 0, existe una raíz única con valor -c/b*
- *En los demás casos, utilizaremos la formula siguiente:*

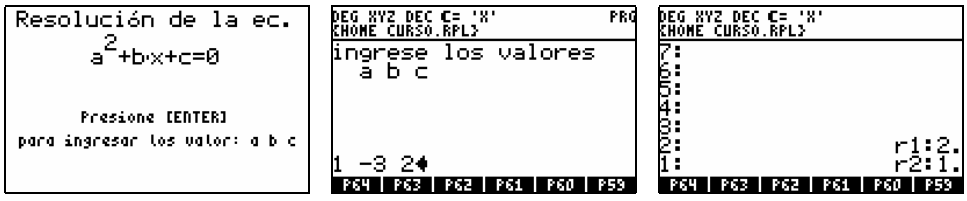
$$x_i = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

La expresión $d = b^2 - 4ac$ *se denomina discriminante.*
Si $d \geq 0$ *entonces hay dos raíces reales*
Si $d < 0$ *entonces hay dos raíces complejas de la forma: x+yi, x-yi.*
Siendo x el valor -b/2a e y el valor absoluto de $\sqrt{\frac{b^2 - 4ac}{2a}}$

P64

```
« ERASE PICT ( # 5d # 2d )
  "Resolución de la ec." 2 →GROB GOR
  PICT ( # 37d # 12d ) 'a^2+b*x+c=0' 0 →GROB GOR
  PICT ( # 33d # 45d )
  "Presione [ENTER]" 1 →GROB GOR
  PICT ( # 6d # 55d )
  "para ingresar los valor: a b c" 1 →GROB GOR
  ( # 0d # 0d ) PVIEW 0 WAIT DROP
  "ingrese los valores
  a b c"
  { "" ( 1 0 ) V }
  INPUT OBJ→ → a b c
  « IF a 0 == b 0 == AND THEN
      CLLCD "La ec. es degenerada" MSGBOX
    ELSE
      IF a 0 == b 0 ≠ AND THEN
        c b / "r1" →TAG
      ELSE
        b SQ 4 a * c * - → d
        « IF d 0 ≥ THEN
            '(-b+√(b^2-4*a*c))/(2*a)' EVAL "r1" →TAG
            '(-b-√(b^2-4*a*c))/(2*a)' EVAL "r2" →TAG
          ELSE
            '-b/2*a' EVAL "x" →TAG
            '(-b-√(b^2-4*a*c))/(2*a)' EVAL "y" →TAG
          END
        »
      END
    END
  »
»
```

Resolvemos para: $x^2 - 3x + 2 = 0$

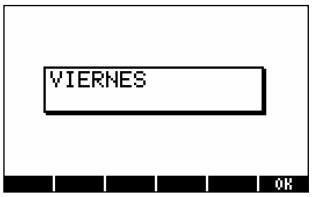


Se dice que una estructura IF (IF..THEN ELSE) esta anidada cuando esta contenida dentro de otra estructura IF o dentro de otra estructura IF…THEN…ELSE. No existe límite en cuanto al nivel de anidamiento. Por ejemplo, una estructura IF con tres niveles de anidamiento tendría el siguiente formato *(Ej. der.)*

```
« IF Condición1 THEN
     Acción A1
  ELSE
    IF Condición2 THEN
       Acción A2
    ELSE
      IF Condición3 THEN
         Acción A3
      ELSE
        Acción B1
        END
      END
    END
  END
»
```

Ejemplo 92. *Dado un número del 1 a 7 escriba el correspondiente nombre del día de la semana.*

```
P65

« "Ingrese # de dia" { "" { 1 8 } V } INPUT OBJ→ → d
  « IF d 1 == THEN
      CLLCD "LUNES" MSGBOX
    ELSE
      IF d 2 == THEN
        CLLCD "MARTES" MSGBOX
      ELSE
        IF d 3 == THEN
          CLLCD "MIERCOLES" MSGBOX
        ELSE
          IF d 4 == THEN
            CLLCD "JUEVES" MSGBOX
          ELSE
            IF d 5 == THEN
              CLLCD "VIERNES" MSGBOX
            ELSE
              IF d 6 == THEN
                CLLCD "SABADO" MSGBOX
              ELSE
                CLLCD "DOMINGO" MSGBOX
              END
            END
          END
        END
      END
    END
  »
»
```



La estructura de selección múltiple es un caso especial de la estructura IF…THE…ELSE...END.

Comandos Condicionales

La HP tiene comandos como IFT y IFTE que reemplazan a IF...THEN...END, IF...THEN...ELSE...END respectivamente y funcionan de la misma manera solo en vez de utilizar cuatro comandos solo se utiliza uno.

IFT

Forma compacta de: **SI...ENTONCES...FIN**

Sintaxis: « *Condición Acción* IFT »

Ejecuta una acción si el valor de la condición es un número real distinto a cero.

IFT

```
« → N
  « N 2 MOD NOT
    "#par" IFT
  »
»
```

IF…THEN…END

```
« → N
  « IF N 2 MOD 0 ==
    THEN "#par"
    END
  »
»
```

La condición puede dar como resultado un número real cualquiera, si la condición fuese cero termina el programa

IFTE

Forma compacta de: **SI...ENTONCES...SINO…FIN**

Sintaxis: « *Condición AcciónA acciónB* IFTE »

Ejecuta una acciónA si el valor de la condición es un número real distinto a cero o otra acciónB, si el valor de la condición es cero

IFTE

```
« → N
  « N 2 MOD NOT
    "# Par" "# Impar"
    IFTE
  »
»
```

IF…THEN…ELSE…END

```
« → N
  « IF N 2 MOD 0 ==
      THEN "# Par"
      ELSE "# Impar"
    END
  »
»
```

Si se lo habrá notado en los ejemplos; son muy similares a las Estructuras condicionales, en realidad son las misma con la única diferencia que distinguen como verdadero cualquier número real (`1...N`), y como falso al cero (`0`).

17.3. Estructuras de selección múltiple

17.3.1. CASE…THEN…END

Traducción: **EN CASO DE...ENTONCES...FIN**

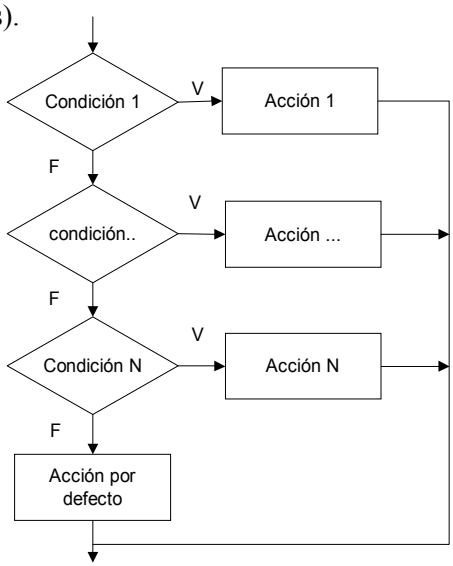
La estructura de selección múltiple CASE permite elegir una ruta de entre varias rutas posibles, usando para ello una variable denominada selector. El selector se compara con una lista de constantes enteras o de carácter C1, C2, ..., Cn para cada una de las cuales hay una acción A1, A2, ..., An

- Si el selector coincide con una constante de la lista, se ejecuta la acción correspondiente a dicha constante.
- Si el selector no coincide con ninguna constante de la lista, se ejecuta la acción por defecto, si es que existe.

Las acciones A1, A2, A3, ..., An pueden ser acciones simples(una sola acción) o acciones compuestas (un conjunto de acciones).

Sintaxis:

```
« ...
  CASE
       Condición 1 THEN Acción 1 END
       Condición 2 THEN Acción 2 END
       .
       .
       Condición N THEN Acción N END
       Acción por defecto (opcional)
  END
  ...
»
```



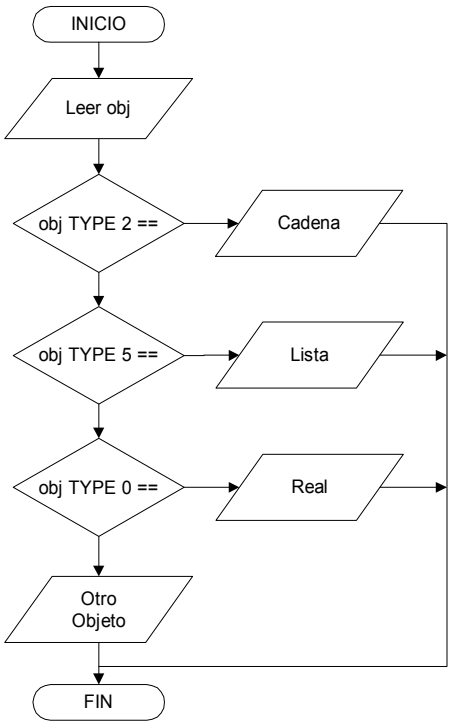
Al ejecutarse CASE, se evalúa la condición 1. Si la condición es verdadera se ejecuta la acción 1. y la ejecución salta a END. Si la condición 1 es falsa la ejecución pasa a la condición 2…

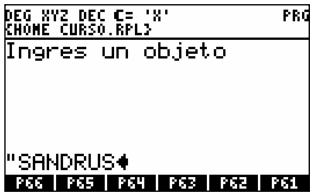
La ejecución dentro la estructura CASE continúa hasta que se ejecuta una condición verdadera o hasta que todas las condiciones dan falso como resultado.

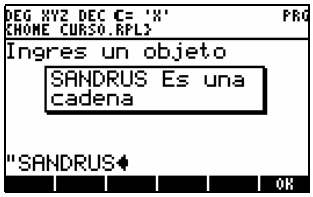
A modo de opción, puede incluir después de la última condición una acción por defecto que se ejecutará si toda las pruebas dan falso. Si una condición es una operación algebraica, da como resultado un número no necesita de EVAL ó →NUM

Ejemplo 93. *Programa que dado un objeto describe el tipo de objeto.*

```
P66

« "Ingres un objeto"
  { "" { 1 0 } V }
  INPUT  OBJ→ → obj
  «
    CASE
    obj TYPE 2 == THEN
    obj →STR " Es una cadena" + END
    obj TYPE 5 == THEN
    obj →STR " Es una lista" + END
    obj TYPE 0 == THEN
    obj →STR " Es un Numero Real" + END
    "ERROR Otro tipo de objeto"
    END MSGBOX
  »
»
```







Ejemplo 94. *Programa que dado un número del 0 a 10 escribe el correspondiente nombre del número.*

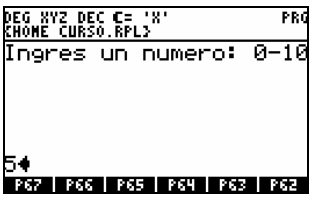
P67

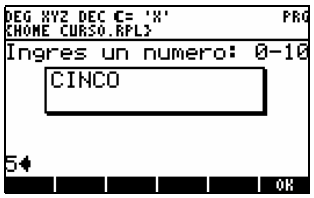
```
« "Ingres un numero: 0-10"
  { "" { 1 0 } V }
  INPUT  OBJ→ → N
  «
    CASE
    N 0 == THEN  "CERO" END
    N 1 == THEN  "UNO" END
    N 2 == THEN  "DOS" END
    N 3 == THEN  "TRES" END
    N 4 == THEN  "CUATRO" END
    N 5 == THEN  "CINCO" END
    N 6 == THEN  "SEIS" END
    N 7 == THEN  "SIETE" END
    N 8 == THEN  "OCHO" END
    N 9 == THEN  "NUEVE" END
    "DATO INCORRECTO"
    END MSGBOX
  »
»
```

Para cada condición se compara si N es igual al número condicionado, si es verdadero se tiene en el nivel-1 de la pila una cadena con el nombre del número. Si ninguno de las condiciones se cumple se ejecuta la acción por defecto.

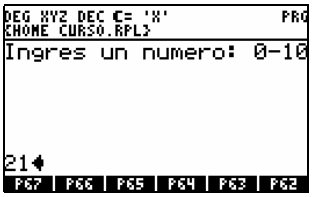
1: 

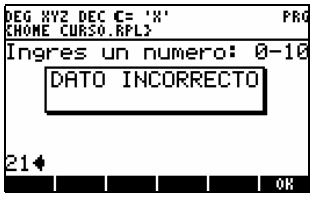
2: 5 ENTER





Si elegimos un número fuera se ejecuta la acción por defecto:



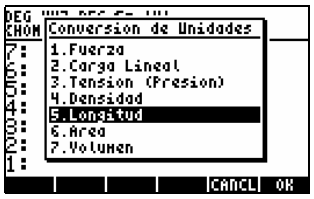


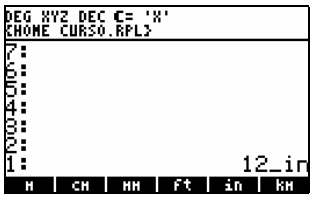
Ejemplo 95. *Programa que trabaja con un cuadro de selección; según el número de ítem se ejecuta dicha operación.*

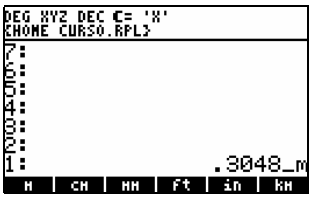
P68

```
« "Conversion de Unidades" { { "1.Fuerza" 1 }
  { "2.Carga Lineal" 2 } { "3.Tension (Presion)" 3 }
  { "4.Densidad" 4 } { "5.Longitud" 5 }
  { "6.Area" 6 } { "7.Volumen" 7. } }  1 CHOOSE DROP → a
  «
    CASE
    a 1 == THEN
    { '1._kN' '1._kip' '1._kgf' '1._tf' '1._N' '1._lbf' }
    TMENU END
    a 2 == THEN
    { '1._kN/m' '1._kgf/m' '1._tf/m' '1._lbf/ft'
    '1._lbf/in' } TMENU END
    a 3 == THEN
    { '1._MPa' '1._kN/m^2.' '1._kN/cm^2.' '1._kgf/cm^2.'
    '1._tf/m^2.' '1._Ksi' '1._psi' '1._N/mm^2.' }
    TMENU END
    a 4 == THEN
    { '1._kN/m^3.' '1._kgf/m^3.' '1._kip/ft^3.'
    '1._lbf/ft^3.' '1._lbf/in^3.' } TMENU END
    a 5 == THEN
    { '1._m' '1._cm' '1._mm' '1._ft' '1._in' '1._km'
    '1._mi' } TMENU END
    a 6 == THEN
    { '1._m^2.' '1._cm^2.' '1._mm^2.' '1._ft^2.'
    '1._in^2.' '1._km^2.' '1._mi^2.' '1._ha' } TMENU END
    a 7 == THEN
    { '1._m^3.' '1._cm^3.' '1._mm^3.' '1._ft^3.'
    '1._in^3.' '1._l' '1._gal' '1._km^3.' '1._mi^3.' }
    TMENU END
    END
  »
»
```

1: [▽] [5] [ENTER]
2: [1] [2] [$\square_E$] [ENTER]
3: [←] [$\square_A$] [ENTER]







17.4. Estructuras de Detección de Errores

La HP reconoce de forma automática gran cantidad de situaciones como las de error y las trata automáticamente como tales en los programas. Un comando con uno o varios argumentos inadecuados produce un error en un programa. Un sepultado fuera de rango puede producir un error. Una condición de cálculo no válida puede producir un error.

Las estructuras de detección de errores permite que los programas detecten (o intercepten) las situaciones de error que, de lo contrario provocarían la suspensión de la ejecución del programa.

17.4.1. IFERR…THEN…END

Traducción: **SI se produce un error en...ENTONCES...FIN**

Sintaxis:

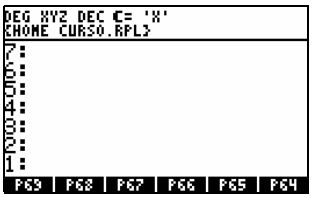
```
« IFERR Acción-detección THEN
        Acción-error
  END
»
```

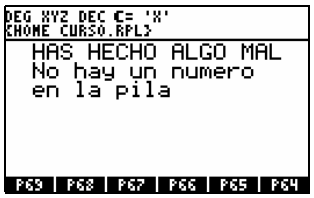
Se ejecuta la *Acción-error* sólo si se produce un error durante la ejecución de la *Acción-detección*. Si se produce un error en la acción de detección se pasa por alto el error, se salta el resto de la acción de detección y la ejecución del programa salta a la acción de error. Si no se produce ningún error en la acción de detección, se salta la acción de error y se reanuda la ejecución después del comando END.

Ejemplo 96. *Programa que muestra el uso de IFERR*

```
P69

« IFERR
  → n
  « n ! »
  THEN
  CLLCD "  HAS HECHO ALGO MAL
  No hay un numero
  en la pila" 3 DISP 2 FREEZE
  END
»
```

Si no existe un número en la pila, se produce un error, al ser evaluado por IF el programa se salta el resto de la acción que se condiciona y se ejecuta la acción de error.

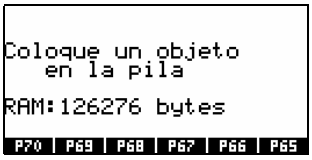


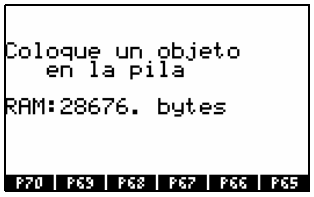


Ejemplo 97. *Programa que muestra el uso de IFERR*

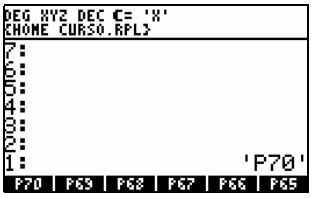
```
P70

« IFERR RCL
  THEN
  END
  IFERR DUP BYTES SWAP DROP '1_bytes' * 'Size' →TAG
  THEN
   CLLCD "RAM:"
   MEM + " bytes" +
   6 DISP
   "Coloque un objeto
   en la pila" 3 DISP 5 WAIT
  END
»
```

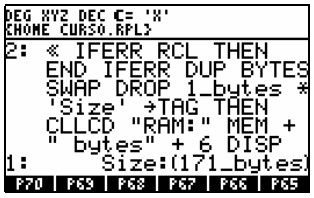
1: ' ▒▒▒
2: ENTER





1: ◀ ▒▒▒ ON
2: ▒▒▒





17.4.2. IFERR…THEN…ELSE…END

Traducción: **SI se produce un error en...ENTONCES...SINO...FIN**

Sintaxis:

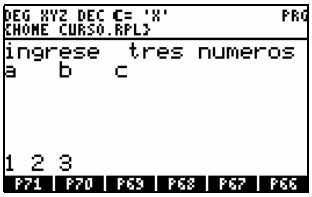
```
« IFERR Acción-detección THEN
        Acción-error
  ELSE Acción-normal
  END
»
```

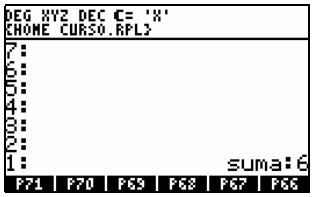
Los comandos de la *Acción-error* se ejecutan sólo si se produce un error durante la ejecución de la *Acción-detección*. Si se produce un error en la acción de detección, se pasa por alto el error, se salta el resto de la acción de detección y la ejecución del programa salta a la acción de error. Si no surge ningún error en la acción de detección, la ejecución salta a la *Acción-normal* a la conclusión de la acción de detección.

Ejemplo 98. *Programa que hace una verificación de ingreso de datos.*

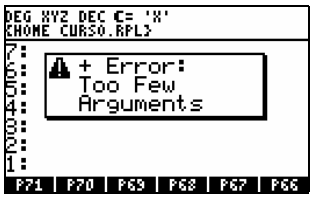
```
P71

« IFERR
"ingrese  tres numeros
a   b   c"
{ "" { 0 0 } V }
INPUT OBJ→
  THEN
  2 DROPN
  "Programa   finalizado" MSGBOX
  ELSE + + "suma" →TAG
  END
»
```

Ejecutamos el programa y ingresamos los valores:





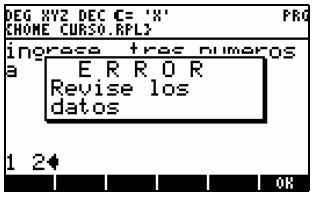
Si por alguna razón el programa se cancela este se interrumpe con la acción del error. Y si ingresamos menos de tres o más números, el programa votara el siguiente error:



El cual también podemos hacer la detección de errores en la ejecución normal modificando el código.

```
P71a

« IFERR
"ingrese  tres numeros
a   b   c"
{ "" { 0 0 } V }
INPUT OBJ→
  THEN
  2 DROPN
  "Programa   finalizado" MSGBOX
  ELSE
  IFERR
   + + "suma" →TAG
   THEN
   "  E R R O R   Revise los datos" MSGBOX
   END
  END
»
```

Si tenemos errores en el ingreso de datos, se cometiese error en la acción normal se tiene:

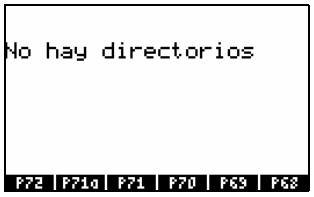


Las estructuras de detección de errores trabajan de similar forma como las estructuras condicionales a diferencia que estos detectan errores y no funciones lógicas.

Ejemplo 99. *Programa que hace una verificación de ingreso de datos.*

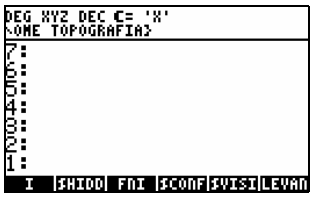
```
P72

« IFERR
  "DIRECTORIOS:" 15 TVARS
  1 CHOOSE
  THEN
  3 DROPN CLLCD
  "No hay directorios"
  3 DISP 0 WAIT DROP
  ELSE
  DROP EVAL
  END
»
```

Ejecutamos el programa:



Es porque en el directorio actual donde estamos trabajando no tenemos directorios creados pero si este ejecutamos en home tendremos:

Si no existen errores en la ejecución de detección se ejecutara la acción normal.



17.4.3. Comandos relacionados con errores

La HP dispone de algunos comandos mas para trabajar con errores y entre ellos tenemos:

17.4.3.1. DOERR

Permite crear un mensaje de error en un recuadro en la parte central de la pantalla, al efectuarse un error en un programa este lo finaliza. También este comando llama a un error específico mediante un número hexadecimal.

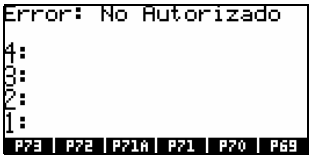
Sintaxis:

```
«...
    "Mensaje" DOERR
 ...
»
```

"Mensaje"	Cadena de Caracteres como mensaje
DOERR	Comando que cancela la ejecución del programa y muestra en pantalla un mensaje de erro.

Ejemplo 100. *Programa que muestra el uso de DOERR*

```
P73

« "Error: No Autorizado"
  DOERR
»
```



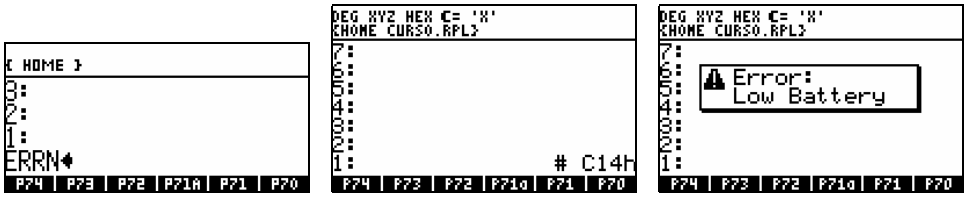
17.4.3.2. ERRN

Este comando devuelve el último error ocasionado en modo hexadecimal.

Ejemplo 101. *Programa que muestra el uso de ERRN*

```
P74
        « ERRN DOERR »
```


Encontramos el último error ocasionado.
Escribimos en la pila: ALPHA E R R N ENTER

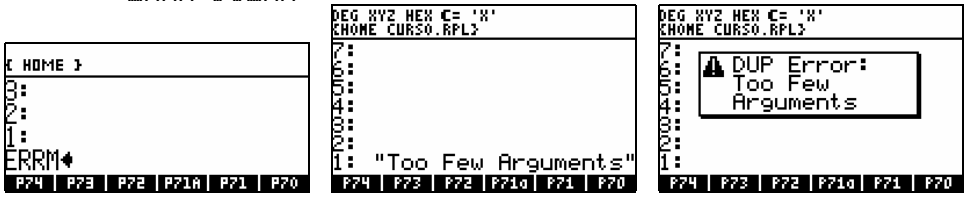


17.4.3.3. ERRM

Este comando devuelve el último error ocasionado en string.

Ejemplo 102. *Programa que muestra el uso de ERRM*

```
P75
      « ERRM DOERR »
```



17.4.3.4. ERR0

Este comando elimina de la memoria temporal el último error ocasionado

17.4.3.5. LAST TARG

Este comando devuelve el último comando en la pila

18. Estructuras de Repetición

Las estructuras de repetición (iterativas o bucles) permiten que un programa ejecute una secuencia de comandos varias veces. Para especificar por cuantas veces ha de repetirse el bucle, utilizase estructuras de repetición definidas. Para utilizar una prueba que determine si hay que repetir o no el bucle utilizase estructuras de repetición indefinidas. Al igual que las estructuras condicionales descritas anteriormente, las estructuras iterativas se construyen con comandos que funcionan solo cuando se utiliza en combinación apropiada con otras.

Para tener acceso a los comandos de las estructuras de repetición ingresamos a PRG en el menú También ingresamos al menú, escribiendo en la pila 23 MENU

18.1. Estructuras de Repetición Definidas

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

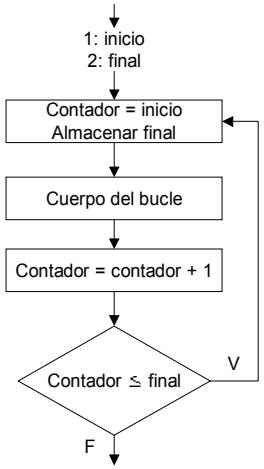
18.1.1. STAR…NEXT

Traducción: **EMPEZAR...SIGUIENTE**

START… NEXT ejecuta la secuencia de comandos de la acción una vez por cada número entre *inicio* y *final*. La acción siempre se ejecuta al menos una vez

Sintaxis:

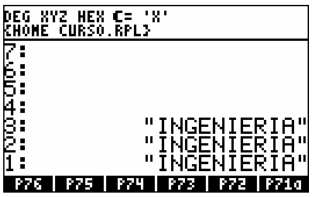
```
« ...
  inicio  final
  START  Acción del bucle  NEXT
  ...
»
```



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

Ejemplo 103. *Programa que imprime un objeto tres veces.*

```
P76

« 1 3
  START "INGENIERIA"
  NEXT
»
```



Ejemplo 104. *Programa que imprime una cadena en la pantalla, varias veces.*

```
P77

« 0 'n' STO
  1 7
  START 1 'n' STO+
  "INGENIERIA" n DISP 1 WAIT
  NEXT
  'n' PURGE
»
```

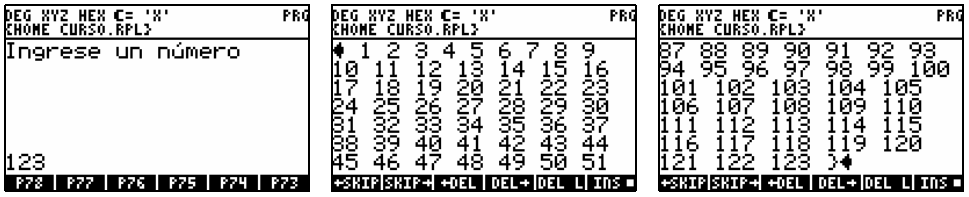
Funcionamiento del programa:

«	*Inicio del programa*
0 'n' STO	*Asigna el valor de 0 a la variable n, y se guarda con el comando STO*
1 7	*Inicio y final de la iteración*
START	*Inicia la iteración determinada*
1 'n' STO+	*Se adiciona 1 a la variable n con STO+*
"INGENIERIA" n DISP 1 WAIT	*Muestra la cadena de caracteres en el nivel-n de la pantalla y se pausa 1 seg. con WAIT*
NEXT	*Finaliza la iteración determinada*
'n' PURGE	*Borra la variable global n*
»	*Fin del programa*

Ejemplo 105. *Programa que crea una sucesión de números en una lista, dado N*

P78

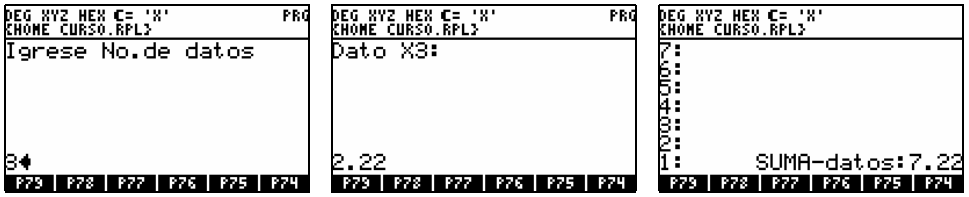
```
« "Ingrese un número"
  { "" { 0 0 } V }
  INPUT OBJ→ → n
  « 0 'a' STO
    0 n 1 -
    START a 1 + 1 'a' STO+
    NEXT
    'a' PURGE n →LIST
  »
»
```



Ejemplo 106. *Programa que halla la suma, y crea la entrada de objetos para N datos.*

P79

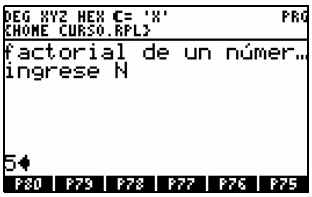
```
« "Igrese No.de datos"
  { "" { 0 0 } V }
  INPUT  OBJ→ 1  → n a
  « 1 n
    START
    "Dato X" a + ":" + "" INPUT OBJ→
    a 1 + 'a' STO
    NEXT
    n →LIST Σ LIST
    "SUMA-datos" →TAG
  »
»
```

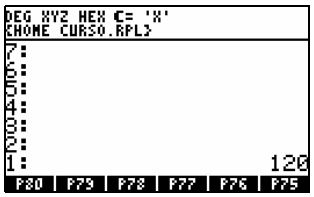


Ejemplo 107. *Programa que calcula el factorial de un número entero.*

P80

```
« "factorial de un número
ingrese N" { "" { 0 0 } V }
  INPUT OBJ→ 1 → n a
  « IF n 0 ==
    THEN 1
    ELSE
      1 n 1 -
      START
      n a * DUP 'a' STO 'f' STO 'n' 1 STO-
      NEXT f 'f' PURGE
    END
  »
»
```

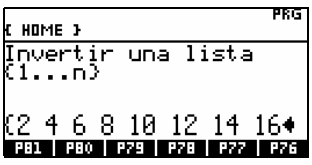


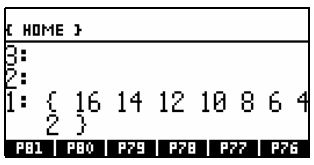


Ejemplo 108. *Programa que invierte una lista de N objetos.*

P81

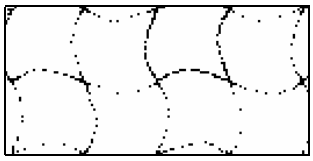
```
« IFERR
"Invertir una lista
{1...n}" { "{}" { 1 2 } }
  INPUT OBJ→ 0 SWAP
  OBJ→ DUP 1 SWAP
  START
  1 - SWAP
  OVER 3 + ROLLD DUP 2 + ROLL 1 +
  OVER 2 + ROLLD
  NEXT DROP →LIST
  THEN
  2 DROPN CLLCD "Programa finalizado" DOERR
  END
»
```

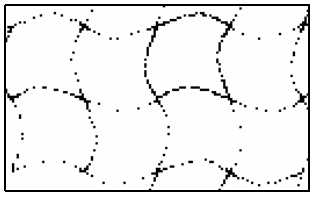




Ejemplo 109. *Programa que muestra el uso START...NEXT con gráficos*

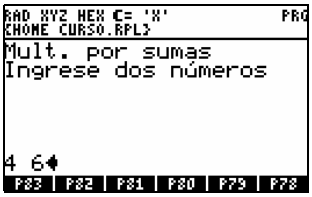
```
P82

« RAD ERASE
  { # 0h # 0h } PVIEW
  3.14 0 0 1 2500 START
  DUP2 R→C C→PX PIXON OVER SIN -
  3 PICK ROT -
  NEXT
»
```

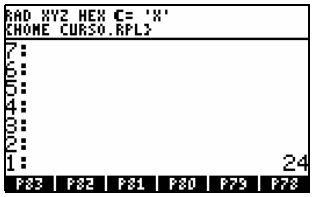




Ejemplo 110. *Programa que realiza la multiplicación de dos números por sumas sucesivas.*

```
P83

« CLEAR
  IFERR
  "Mult. por sumas
Ingrese dos números"
  { "" { 1 0 } }
  INPUT OBJ→ → a b
  « 0 1 a START
    b +
    NEXT
  »
  THEN
  DROP2
  "Prg.finalizado" DOERR
  END
»
```





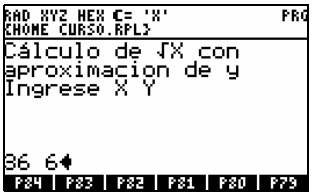
Ejemplo 111. *Prg. que calcula la raíz cuadrada con una aproximación de y*

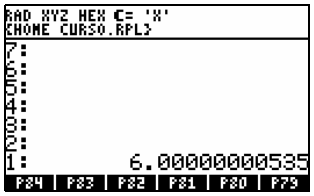
P84

```
« "Cálculo de √X con
aproximacion de y
Ingrese X Y"
  { "" { 1 0 } }
  INPUT OBJ→ → V I
  « 1 DUP I
    START
    V OVER / + .5 *
    NEXT
  »
»
```

$$U_0 = 1$$

$$U_{n+1} = \frac{1}{2}\left(U_n + \frac{a}{U_n}\right)$$

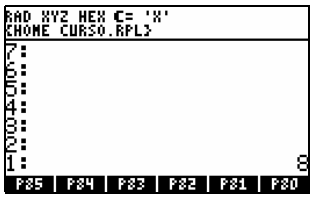




Ejemplo 112. *Programa que realiza la potenciación por sumas sucesivas.*

P85

```
« "Pot.por adición X^Y
Ingrese X Y"
  { "" { 1 0 } }
  INPUT OBJ→ → a b
  « 1 DUP b
    START
      0 1 a
      START
      OVER +
      NEXT SWAP DROP
    NEXT
  »
»
```



Ejemplo 113. *Programa que calcula el valor de π por el método de Montecarlo.*

P86

```
« IFERR
CLLCD "Cálculo de π
Metodo de Montecarlo
con N iteraciones N>0
# de Iteraciones" 2 DISP 2 WAIT
"# de Iteraciones
N=?"
  { "" { 1 0 } }
  INPUT OBJ→ → n
  « IF n 0 > THEN
      1 DUP n
      START
      IF RAND SQ RAND SQ + 1 <
      THEN 1 +
      END
      NEXT n / 4 *
    ELSE
    "E R R O R    N>0" DOERR
    END
  »
  THEN
  CLEAR CLLCD "Prg.finalizado" DOERR
  END
»
```

Método de Montecarlo:

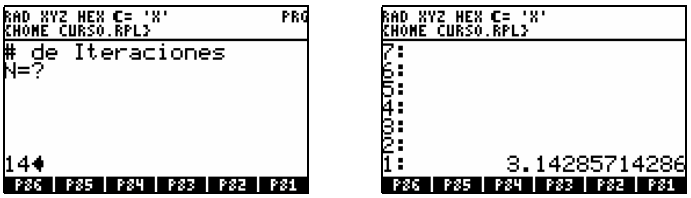
Cálculo de π alrededor de un contorno

$$C\left(x^2+y^2<1\right)$$

Donde:

$$\pi = 4\left(\frac{C}{n}\right)$$

x e y son aleatorios



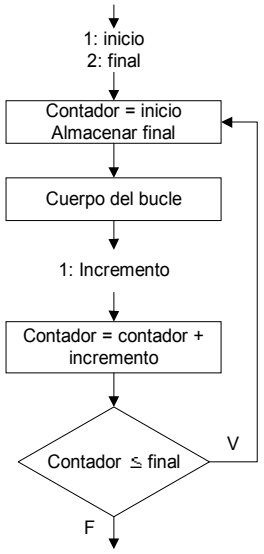
18.1.2. STAR…STEP

Traducción: **EMPEZAR...PASO**

START… STEP ejecuta la secuencia de comandos de la acción una vez por cada número entre *inicio* y *final*. Solo que el programa especifica el valor de incremento para el contador en vez de incrementarlo por 1. La acción del bucle se ejecuta siempre al menos una vez.

Sintaxis:

```
«
  ▪ ▪ ▪
  inicio  final
  START  Acción del bucle
  incremento  STEP
  ▪ ▪ ▪
»
```



START toma dos números (inicio y final) de la pila y los almacena como los valores, inicial y final del contador de bucle. A continuación, se ejecuta la acción del bucle. STEP toma el valor de incremento de la pila e incrementa el contador por ese valor. Si el argumento de STEP es una operación algebraica o un nombre, da automáticamente como resultado un número.

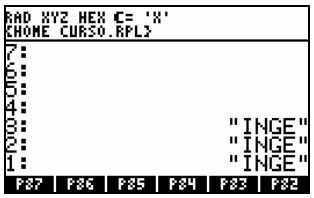
El valor de incremento puede ser positivo o negativo. Si es positivo se vuelve a ejecutar el bucle si el contador es menor que o igual a final. Si el valor del incremento es negativo, se ejecuta el bucle si el contador es mayor que o igual a final. De lo contrario, se reanuda la ejecución con STEP, que viene a continuación.

Para tener acceso a los comandos de esta estructura ingresamos a PRG en el menú [Exec] [Strng] También ingresamos al menú, escribiendo en la pila, `26 MENU`

Ejemplo 114. *Programa que imprime una cadena de caracteres.*

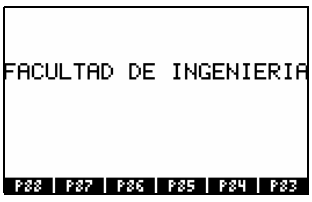
```
P87

« 1 3
  START
  "INGE" 1
  STEP
»
```

«	*Inicio del programa*
1 3	*Inicio y final de la iteración*
START	*Inicia la iteración determinada*
"INGE"	*Cadena de caracteres puesto en la pila*
1	*Número de Incremento*
STEP	*Finaliza la iteración determinada*
»	*Fin del programa*



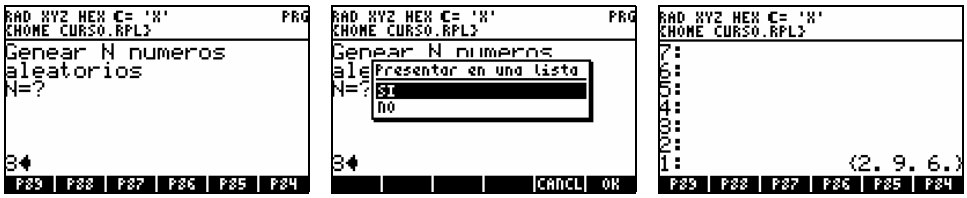
Ejemplo 115. *Programa que imprime en la pantalla una cadena iterativamente.*

```
P88

« 3 1
  START
  CLLCD "FACULTAD DE INGENIERIA"
  4 DISP 1 WAIT
  "   INGENIERIA CIVIL"
  4 DISP 1 WAIT -1
  STEP
»
```



Ejemplo 116. *Programa genera N números aleatorios, dado N.*

```
P89

« IFERR
"Genear N numeros
aleatorios
N=?"
  { "" { 1 0 } V }
  INPUT OBJ→ → n
  « 1 n
    START
    RAND 10 * IP 1
    STEP
    "Presentar en una lista"
    { { "SI" 1 }
    { "NO" 2 } }
    1 CHOOSE DROP → s
    «
      IF s 1 ==
      THEN
      n →LIST
      END
    »
  »
  THEN
  DROP2 "Programa Finalizado" DOERR
  END
»
```



Ejemplo 117. *Programa que construye un vector 1xN, dado N.*

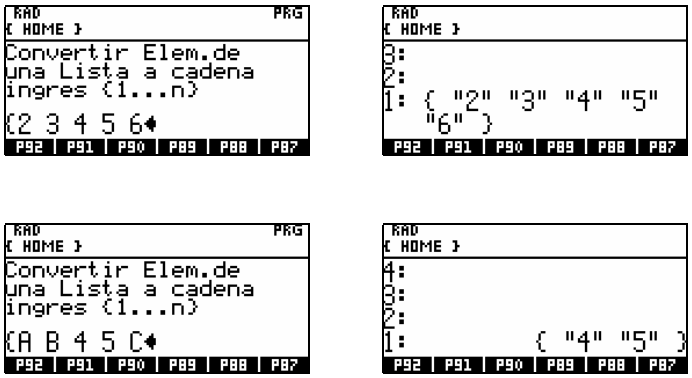
```
P90

« "Contruir un vector
de N elementos
N=?"
  { "" { 1 0 } V }
  INPUT OBJ→ 0 0 → n a u
  « { n } 0 CON
    'a' STO n 1
    START 'u' INCR
    'u' STO
    "Elemento No." u + ""
    INPUT OBJ→ 'a(u)' STO -1
    STEP a
  »
»
```

Ejemplo 118. *Programa que realiza una interpolación.*

```
P91

« IFERR
"Ingrese El intervalo
Xi...Xf ej: 1 7"
  { "" { 1 0 } V }
  INPUT OBJ→
  "Valor p/interpolar
  dx  ej: .5"
  { "" { 1 0 } V }
  INPUT OBJ→ → xi xf dx
  « xi DUP xf
    START
    DUP dx + dx
    STEP DROP xf xi - dx / ABS 1 + →LIST
  »
  THEN
  DROP2 "Programa Finalizado" DOERR
  END
»
```

Ejemplo 119. *Programa que convierte los elementos de una lista en caracteres.*

```
P92

« "Convertir Elem.de
una Lista a cadena
ingrese {1...n}"
  { "{}" { 1 2 } V }
  INPUT OBJ→ 0 SWAP
  OBJ→ DUP 1 SWAP
  START
  1 - SWAP
    IF DUP TYPE 0 SAME THEN
      →STR OVER DUP 4 +
      PICK + 3 + ROLLD DUP
      2 + ROLL 1 + OVER
      2 + ROLLD
    ELSE
      DROP
    END 1
  STEP DROP →LIST
»
```



Ejemplo 120. *Programa que construye una matriz MxN*

P93

```
« "Construir una Matriz
mxn: Ingrese m n"
  { "" { 1 0 } V }
  INPUT OBJ→
  0 0 0 → m n a u v
  « { m n } 0 CON
    'a' STO m 1
    START 'u' INCR
    'u' STO n 1
      START 'v' INCR
      'v' STO "Elemento " u + "x"
      + v + "" INPUT OBJ→ 'a(u,v)' STO -1
      STEP 0 'v' STO -1
    STEP a
  »
»
```

INCR *Este comando incrementa una variable especificada, toma un nombre de variable local o global como argumento y luego incrementa en una unidad.*

Programa	Comentario

`« 1 → a`
`  « 'a' INCR`
`  'a' STO a`
`  »`
`»`
	El valor de `'a'` *es igual a* `1`
	`INCR` *incrementa con una unidad a la variable* `'a'` *y luego se guarda con* `STO`
	El nuevo valor de `'a'` *es* `2`

DECR *Este comando decrementa una variable especificada, toma un nombre de variable local o global como argumento y luego reduce una unidad.*

Programa	Comentario

`« 6 → b`
`  « 'b' DECR`
`  'b' STO b`
`  »`
`»`
	El valor de `'b'` *es igual a* `6`
	`DECR` *resta una unidad a la variable* `'b'`
	se guarda el nuevo valor con `STO`
	El nuevo valor de `'b'` *es* `5`

La variable debe contener un número real.

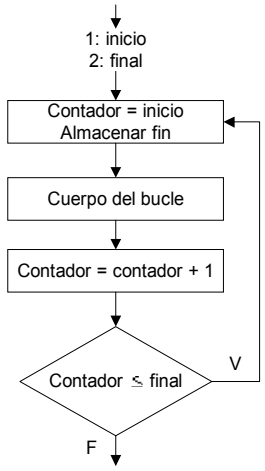
18.1.3. FOR…NEXT

Traducción: **PARA contador...SIGUIENTE**

FOR…NEXT ejecuta el segmento del programa acción del bucle una vez por cada número entre inicio y final, utilizando el contador de la variable local como contador de bucle. Puede utilizarse esta variable en la acción del bucle. La acción del bucle se ejecuta siempre a menos una vez.

Sintaxis:

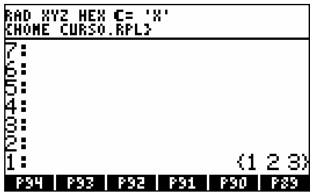
```
« ...
  inicio  final
  FOR  Contador
  Acción del bucle
  NEXT
  ...
»
```



FOR toma inicio y final de la pila como los valores inicial y final para el contador de bucle y crea el contador de la variable local como contador de bucle. A continuación se ejecuta la acción del bucle el contador puede aparecer dentro de la acción del bucle. NEXT incrementa el nombre contador por uno y, a continuación, comprueba si su valor es menor que o igual a final. Si loes, se repite la acción del bucle (con el nuevo valor contador) de lo contrario, se reanuda la ejecución con NEXT, que viene a continuación. Cuando no existe el bucle, se elimina el contador.

Ejemplo 121. *Programa que genera una lista de tres objetos.*

```
P94

« {} 1 3
  FOR i i +
  NEXT
»
```

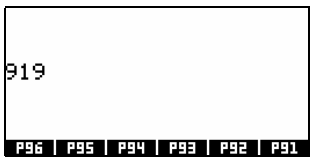


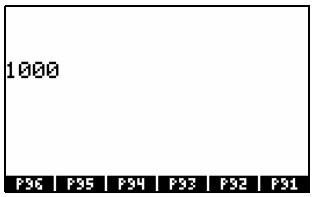
Ejemplo 122. *Programa que genera una secuencia de pitidos.*

```
P95

« 1 10
  FOR f
  100 f * .5 BEEP
  NEXT
»
```

Ejemplo 123. *Programa que imprime en la pantalla los números del 1 al 1000*

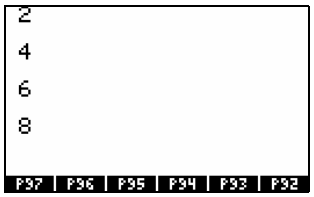
```
P96

« CLLCD
  1 1000
  FOR i
  i 4 DISP 3 FREEZE
  NEXT
»
```





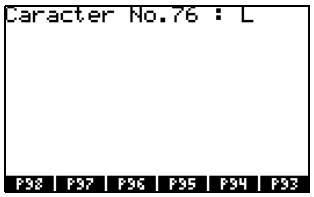
Ejemplo 124. *Programa que calcula los números pares entre 1 y 8*

```
P97

« 1 8 CLLCD
  FOR i
  IF i 2 MOD 0 == THEN
  " " i + i 1 - DISP 7 FREEZE
  ELSE
  END
  NEXT
»
```



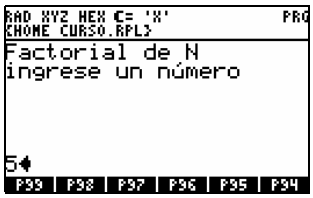
Ejemplo 125. *Programa que muestra en pantalla todos los caracteres.*

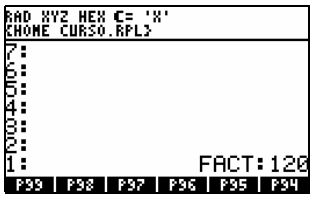
```
P98

« CLLCD 0 225
  FOR i
  "Caracter No." i + " : " +
  i CHR + 1 DISP 0.25 WAIT
  NEXT
»
```



Ejemplo 126. *Programa que calcula el factorial de un número*

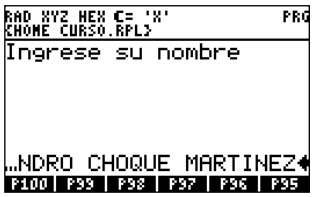
```
P99

« "Factorial de N
ingrese un número"
  { "" { 1 0 } V }
  INPUT OBJ→ → n
  « IF n 0 == THEN
    1
    ELSE
      1 1 n
      FOR i
      i *
      NEXT
    END
  » "FACT" →TAG
»
```





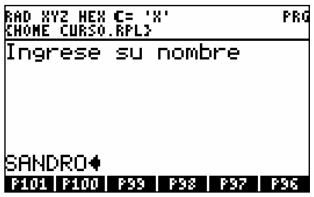
Ejemplo 127. *Programa que simula una máquina de escribir.*

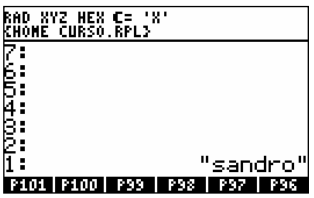
```
P100

« "Ingrese su nombre"
  { "" { 1 2 } « }
  INPUT DUP SIZE → T n
  « CLLCD "" 1 n
    FOR j
    T HEAD T TAIL 'T' STO
    + DUP 'N' STO
    'N' EVAL 1 DISP .6 WAIT
    NEXT
  » 'N' PURGE CLEAR
»
```



Ejemplo 128. *Programa que convierte de mayúsculas a minúsculas e viceversa.*

```
P101

« "Ingrese su nombre"
  { "" { 1 2 } « }
  INPUT DUP SIZE → T n
  « 1 n
    START T HEAD → L
      «
        CASE L NUM 10 ==
          THEN ""
          END L NUM 32 ==
          THEN " "
          END L NUM 65 <
          THEN
          "Solo letras del ABC.."
          DOERR
          END L NUM 97 <
          THEN L NUM 32 + CHR
          END L NUM 123 <
          THEN L NUM 32 - CHR
          END
        END
      » T TAIL 'T' STO
    NEXT
  » DEPTH → R
  « 2 R
    START +
    NEXT
  »
»
```





Ejemplo 129. *Programa que trabaja con gráficos.*

```
P102

« ( # 0h # 0h ) PVIEW
  ERASE PICT ( # 6h # 5h )
  "FACULTAD NACIONAL DE INGENIERIA" 1 →GROB REPL
  PICT ( # 3h # 10h )
  "CURSO DE PROGRAMACION" 2 →GROB REPL
  PICT ( # 25h # 19h )
  "en USER-RPL" 2 →GROB REPL
  PICT ( # 5h # 28h )
  "Expositor:" 1 →GROB REPL
  PICT ( # 24h # 32h )
  "Sandro Choque Martínez" 1 →GROB REPL
  ( # 0h # 0h ) ( # 82h # 3Fh ) BOX 1 WAIT
  # 0h # 83h → x y
  « 1 2
    START
    ( # 0h # 0h ) ( # 83h # 0h ) TLINE
      1 63
      FOR n
      x n R→B 2 →LIST y n R→B
      2 →LIST TLINE
      NEXT
    NEXT
  »
»
```

131 × 64

131 × 80

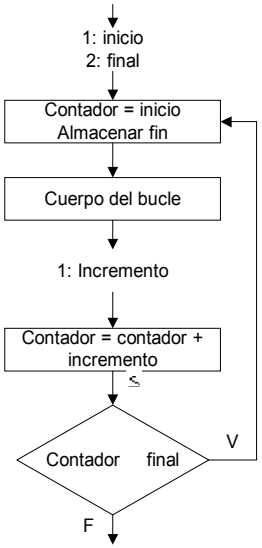
18.1.4. FOR…STEP

Traducción: **PARA contador...incremento PASO**

FOR…STEP ejecuta el segmento del programa acción del bucle una vez por cada número entre inicio y final, sólo que el programa especifica el valor del incremento del contador, en vez de incrementar por 1. La acción del bucle se ejecuta siempre a menos una vez.

Sintaxis:

```
«
  ...
  inicio  final
  FOR  Contador
  Acción del bucle
  Incremento
  STEP
  ...
»
```



FOR toma inicio y final de la pila como los valores inicial y final para el contador de bucle y crea el contador de la variable local como contador de bucle. A continuación se ejecuta la acción del bucle el contador puede aparecer dentro de la acción del bucle. STEP toma el valor del incremento de la pila e incrementa el contador por ese valor. Si el argumento de STEP es una operación algebraica o un nombre, da automáticamente como resultado un número

El valor del incremento puede ser positivo o negativo, Si el incremento es positivo, se vuelve a ejecutar el bucle si el contador es menor que o igual a inicio. Si el incremento es negativo, se ejecuta el bucle si el contador es mayor que o igual a final. DE lo contrario, se elimina el contador y se reanuda la ejecución después de STEP

Ejemplo 130. *Programa que imprime en pantalla los números de 0 al 2000*

```
P103

« 0 2000
  FOR i
  CLLCD i 3 DISP 3 FREEZE 10
  STEP
»
```

«	*Inicio del programa*
0 2000	*Inicio y final de la iteración*
FOR	*Inicia la iteración determinada*
i	*Variable contador*
CLLCD	*Limpia la pantalla*
i	*Coloca enl a pila el valor de contador*
3 DISP 3 FREEZE	*Se muestra el valor del contador*
10	*Valor del incremento para el contador*
STEP	*Finaliza la iteración determinada*
»	*Fin del programa*

Ejemplo 130. *Programa que genera una secuencia de pitidos*

```
P104

« 100 1000
  FOR f 1100 f - .5 BEEP 100
  STEP
»
```

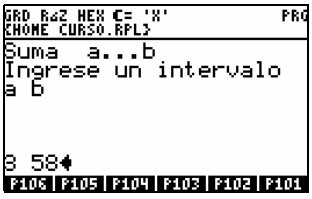
Ejemplo 131. *Programa que genera una secuencia de pitidos*

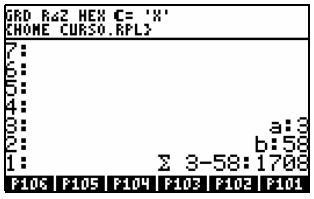
```
P105

« 100 1000
  FOR f f .1 BEEP 50 STEP
  1000 100 FOR f f .1 BEEP -50
  STEP
»
```

Ejemplo 132. *Programa que calcula la sumatoria de un intervalo.*

P106

```
« CLEAR
"Suma  a...b
Ingrese un intervalo
a b"
{ "" { 1 0 } V }
  INPUT OBJ→ → a b
  « 0 a b
    FOR c c + 1
    STEP → d
    « a "a" →TAG b
      "b" →TAG d "Σ " a
      →STR "-" b →STR +
      + + →TAG
    »
  »
»
```

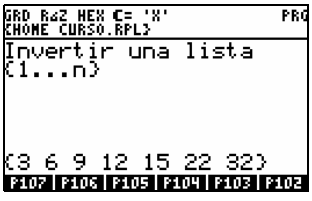




Ejemplo 133. *Programa que invierte una lista*

P107

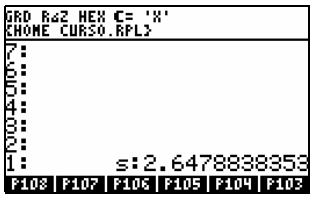
```
« "Invertir una lista
{1...n}"
  { "{} " { 1 2 } V }
  INPUT OBJ→ DUP SIZE → l t
  « { } t 1
    FOR n l n GET + -1
    STEP
  »
»
```



Ejemplo 134. *Programa que halla la suma de la serie:*

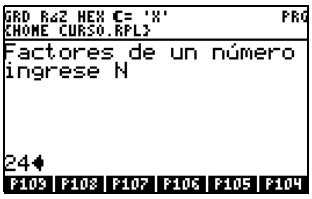
$$s = 1 + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + ... + \frac{1}{N}$$

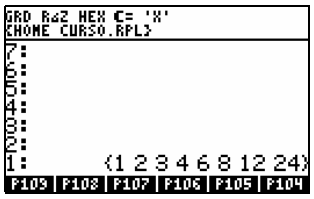
```
P108

« "Suma de la serie
1+1/3+1/5+1/7+...+1/N"
  { ":N:" { 1 4 } V }
  INPUT OBJ→ DTAG → N
  « IF N 3 > THEN
      1 N
      FOR j 1 j / 2
      STEP
      DEPTH 2 SWAP
      START +
      NEXT "s" →TAG
    ELSE
    "ERROR  N<3" DOERR
    END
  »
»
```



Ejemplo 135. *Programa que calcula los factores primos de un número.*

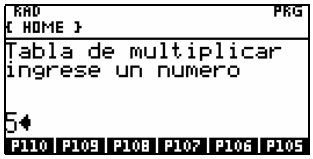
```
P109

« "Factores de un número
ingrese N"
  { "" { 1 0 } V }
  INPUT OBJ→ → n
  « n DUP 2 SWAP
    FOR a DUP a / DUP IP SAME → b
    «
      IF b 1 == THEN
      a SWAP
      END
    » 1
    STEP DROP 1 DEPTH ROLLD DEPTH →LIST
  »
»
```

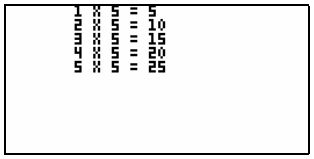


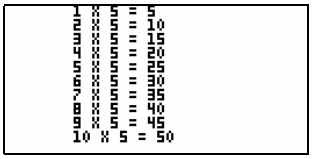


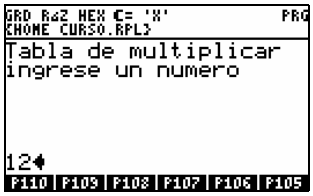
Ejemplo 136. *Programa que imprime en pantalla una tabla de multiplicar*

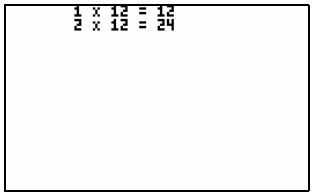
```
P110

« IFERR
"Tabla de multiplicar
ingrese un numero"
  { "" { 1 0 } V }
  INPUT OBJ→ → n
  « ERASE { # 0h # 0h } PVIEW
    1 10 FOR x PICT x " x " + n + " = "
    + n x * + 1 →GROB
    # 1Eh x 1 - 6 * R→B 2 →LIST
    SWAP GOR .5 WAIT 1
    STEP
  » { } PVIEW
  THEN
  DROP2 "Programa finalizado" DOERR
  END
»
```

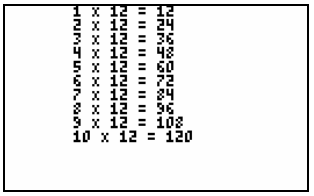












18.2. Estructuras de Repetición Indefinidas

La estructuras de repetición indefinida no especifican el número de veces que ha de repetirse el código, más bien están sujetas a una condición lógica, es decir que se ha de repetir el código hasta que se cumpla dicha condición.

Para tener acceso a los comandos de las estructuras de repetición indefinidas ingresamos a PRG en el menú [PRG] También ingresamos al menú, escribiendo en la pila `23 MENU`

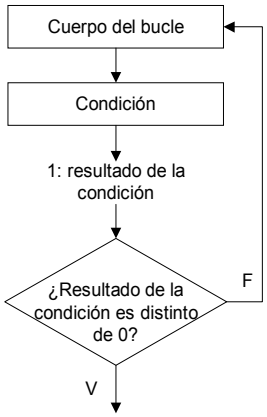
18.2.1. DO…UNTIL…END

Traducción: **HACER...HASTA que…FIN**

DO…UNTIL…END ejecuta la secuencia de la acción del bucle de forma repetitiva basta que la condición del bucle devuelve un resultado verdadero (distinto a cero). Como la condición del bucle se ejecuta después de la acción, esta se ejecuta siempre al menos una vez.

Sintaxis:

```
« ...
  DO
  Acción del bucle
  UNTIL
  Condición del bucle
  END
  ...
»
```



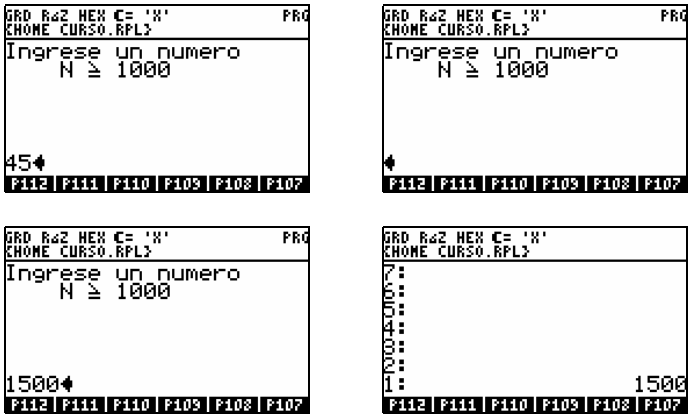
DO inicia la ejecución de la acción del bucle. UNTIL marca el final de la acción. La secuencia de la acción del bucle deja un resultado de prueba en la pila. END retira el resultado de prueba de la pila. Si su valor es cero, la Acción del bucle se vuelve a ejecutar; de lo contrario, se reanuda la ejecución después de END. Si el argumento de END es una operación algebraica o un nombre, da automáticamente como resultado un número.

Ejemplo 137. *Programa que imprime un número hasta que se presione una tecla.*

```
P111

« CLLCD 0
  DO
    DUP 3 DISP 1 +
  UNTIL KEY
  END DROP
»
```

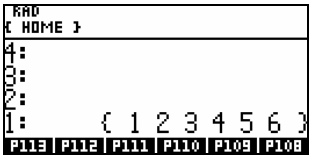
Ejemplo 138. *Programa que pide el ingreso de un número hasta que este sea mayor o igual a 1000*

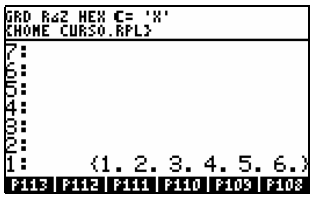
```
P112

« STD
  DO CLEAR
    "Ingrese un numero
    N ≥ 1000"
    { "" { 1 0 } V }
    INPUT OBJ→
  UNTIL DUP 1000 ≥
  END
»
```



Ejemplo 139. *Programa que crea una lista con número aleatorios*

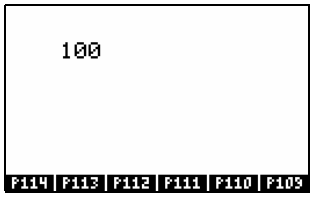
```
P113

« { }
  DO 6 RAND *
  FLOOR 1 + DUP2 POS
  NOT { + } { DROP }
  IFTE
  UNTIL DUP SIZE 6 ==
  END SORT
»
```

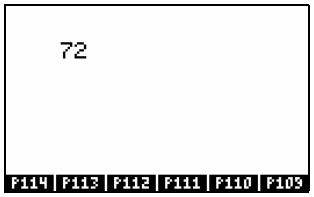


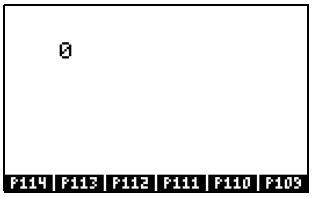


Ejemplo 140. *Programa que muestra en pantalla un decremento de 100 a 0*

```
P114

« IFERR
  100 'A' STO CLLCD
  "    " A + 3 DISP .20 WAIT
  DO CLLCD
    "    " 'A' DECR +
    3 DISP .20 WAIT
    UNTIL A 0 ==
    END 'A' PURGE
    THEN
    'A' PURGE
    "Programa finalizado" DOERR
  END
»
```

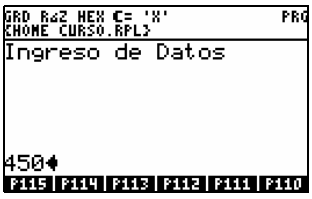


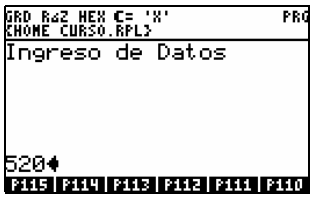


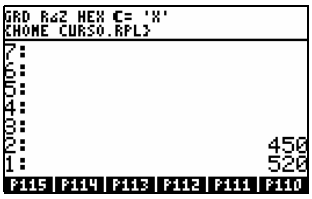


Ejemplo 141. *Programa que enseña como crear un ingreso de datos en forma infinita.*

```
P115

« IFERR
  "Ingreso de Datos"
  { "" { 1 0 } V }
  INPUT OBJ→
  DO
    CLLCD
    "Desea ingresar
    mas datos" 1 DISP
    { "SI" "" "" "" "" "SALIR" }
    TMENU -1 WAIT  { 11.1 16.1 }
    SWAP POS 1 -
      IF DUP THEN
        1 ==
      ELSE
        2 MENU
        "Ingreso de Datos"
        { "" { 1 0 } V }
        INPUT OBJ→ SWAP
      END
    UNTIL
    END 2 MENU
  THEN
  DROP2 "Programa finalizado" DOERR
  END
»
```



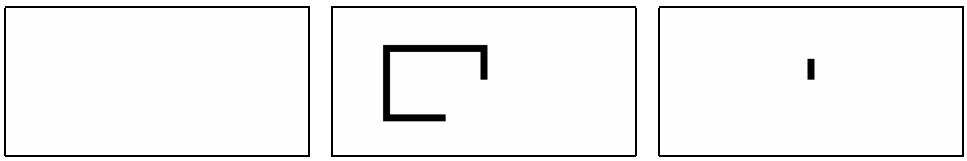




Ejemplo 142. *Programa que enseña el manejo de gráficos.*

P116

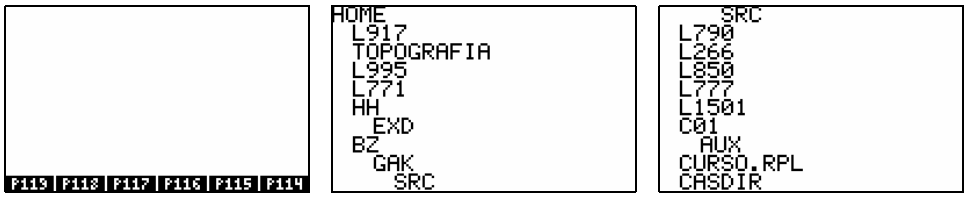
```
« { { 3 0 } { -3 0 } { 0 3 } { 0 -3 } }
  { 36 34 35 25 } { 64 31 } → mov tec np
  « ERASE { # 0d # 0d } PVIEW
    DO 0 WAIT IP → k
    « IF tec k POS DUP
      THEN mov SWAP GET np ADD DUP 'np' STO R→B
      PICT SWAP # 3d # 3d BLANK NEG REPL
      ELSE DROP
      END
    »
    UNTIL 0 END
  »
»
```



Ejemplo 143. *Programa que muestra todos los directorios y subdirectorios de HOME, en forma de un árbol.*

P117

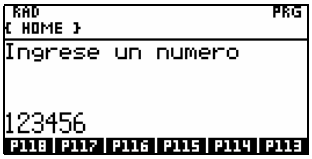
```
« CLLCD PATH HOME { :0: "HOME" } '$2' STO
  « 1 + 15 TVARS 1
    IF OVER SIZE
    THEN
      DO GETI DUP 2 DISP DUP 5 PICK →TAG PATH HOME
      SWAP $2 SWAP + '$2' STO EVAL EVAL 3 PICK $1
      UNTIL DUP 1 ==
      END
    END 3 DROPN
  » '$1' STO → r
  « 0 $1 HOME $2 { $1 $2 } PURGE r EVAL CLLCD DUP SIZE 8
* R□B # 83h SWAP BLANK SWAP :0: "" + 1
    DO GETI OBJ→ OBJ→ 9 * R→B 3 PICK
    2 - 8 * R→B 2 →LIST 5 ROLL 3 ROLLD SWAP DUP →TAG
    OBJ→ SWAP DROP 2 →GROB REPL 3 ROLLD
    UNTIL DUP 1 ==
    END DROP2 PICT STO { } PVIEW PICT PURGE
  »
»
```

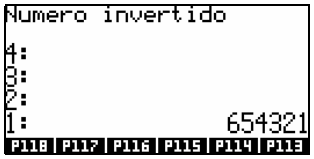


Si HOME no tiene directorios se muestra:

Ejemplo 144. *Programa que invierte cualquier numero entero.*

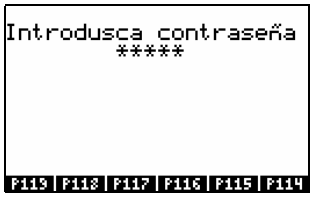
```
P118

« "Ingrese un numero"
  { "" { 1 0 } V }
  INPUT OBJ→ DUP
  IF TYPE 0 == THEN
  0 → N NI
  « DO
    N 10 MOD
    N 10 / IP
    'N' STO
    NI 10 * +
    'NI' STO
    UNTIL
    N 0 ==
    END NI "Numero invertido" DOERR
  »
  END
»
```

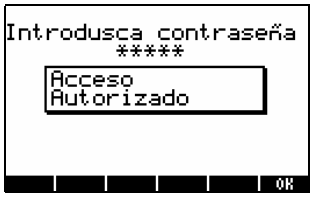




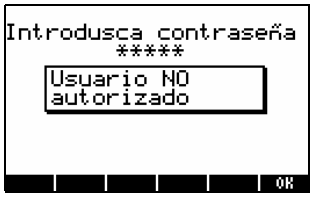
Ejemplo 145. *Programa que pide una contraseña para ingresar.*

```
P119

« CLLCD "Introdusca contraseña
        *****"
  2 DISP
  1 5
  START
    DO
    UNTIL KEY
    END
    1000 .1 BEEP
  NEXT
  @ 11=A 12=B 13=C 14=D 15=E
  IF 5 →LIST ( 11. 12. 13. 14. 15. ) == THEN
    1 6
    FOR i i 100 * .2 BEEP
    NEXT
    "Acceso Autorizado" MSGBOX
    "Prg/grob a ejecutar"
  ELSE
    "Usuario NO autorizado" 500 .3 BEEP MSGBOX
  END
»
```





No se puede acceder si la contraseña no coincide. {A B C D E}



Caso especial de la estructura DO…

Este caso se da, cuando se tienen dos acciones a realizar, como se muestra en la siguiente sintaxis:

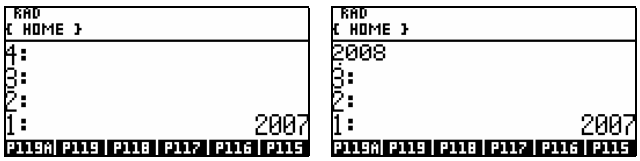
Sintaxis:

```
« ...
  DO
  Acción 1
  UNTIL Número ó Condición
  IF Número ó Condición NOT THEN
  Acción 2
  END
  END
  ...
»
```

Cuando hay varias acciones y condiciones es muy complicado trabajar con la estructura DO, la única manera es introduciendo el comando IF dentro del comando DO y la condición a realizar seria lo inverso (con NOT) *a la condición que realiza con DO*

Ejemplo. *Escribamos un programa que tome un numero de la pila y lo incremente de uno en uno sucesivamente hasta que se presiones la tecla A, solamente* ▭A

```
« DO
  1 +  DUP 3 DISP .25 WAIT
  UNTIL
    IF KEY DUP THEN
      SWAP 11 ≠ NOT IFT
    END
  END
»
```



El programa se terminará si solamente presionamos ▭A *y* ON

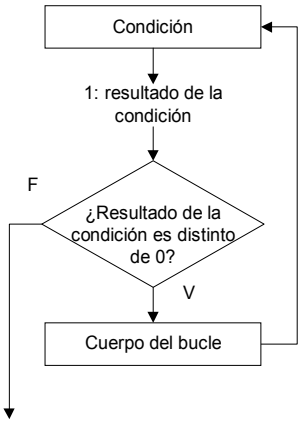
18.2.2. WHILE…REPEAT…END

Traducción: **MIENTRAS...REPETIR…FIN**

WHILE…REPEAT…END evalúa de forma repetida la condición del bucle y ejecuta la secuencia de la acción del bucle si la prueba es verdadera. Como la condición se ejecuta antes de la acción, ésta no se ejecuta si la condición es inicialmente falsa.

Sintaxis:

```
« ...
  WHILE
  Condición del bucle
  UNTIL
  Acción del bucle
  END
  ...
»
```



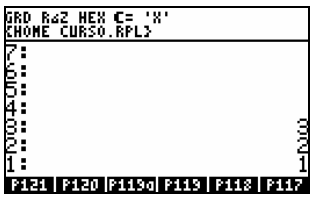
WHILE inicia la ejecución de la condición, la cual devuelve un resultado de prueba a la pila. REPEAT toma el valor de la pila. Si el valor es distinto a cero, continúa la ejecución con la acción del bucle; de lo contrario se reanuda la ejecución después de END. Si el argumento de REPEAT es una operación algebraica o un nombre, da automáticamente como resultado un número.

Ejemplo 146. *Programa que calcula los factores de un número*

```
P120

« WHILE DUP 2 MOD 0 ==
  REPEAT
  2 / DUP
  END DROP
»
```

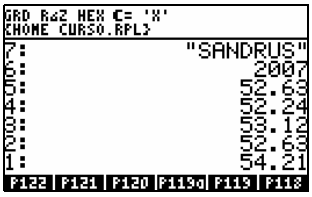
Ejemplo 147. *Programa que imprime tres números en la pila.*

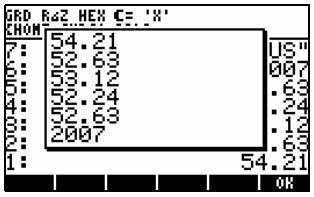
```
P121

« 3 → i
  « WHILE i 0 >
    REPEAT
    i 'i' 1 STO-
    END
  »
»
```



Ejemplo 148. *Programa que muestra en un cuadro los objetos de los niveles menores a 6 de la pila, se puede aplicar este método para mostrar resultados*

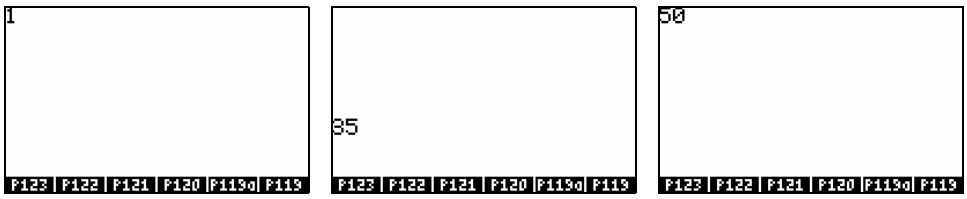
```
P122

« ""
  WHILE DEPTH 1 ≠
  REPEAT SWAP
    WHILE DUP SIZE 14 <
    REPEAT "  " +
    END +
  END MSGBOX
»
```





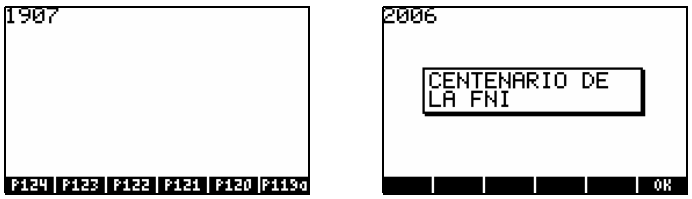
Ejemplo 149. *Programa que imprime en pantalla los números de 1 al 50*

```
P123

« 0 1 → n d
  « WHILE n 50 <
    REPEAT CLLCD
    "" 'n' INCR +
    d DISP .25 WAIT 'd' INCR DROP
      IF d 8 == THEN
        1 'd' STO
      END
    END
  »
»
```



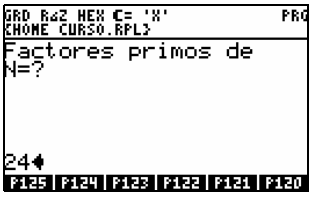
Ejemplo 150. *Programa que imprime en pantalla los números de 1906 hasta que el contador* a *tenga el valor de 2006, para reasignar esta misma variable a cero.*

```
P124

« 1906 → a
  « WHILE a 0 >
    REPEAT
    'a' 1 STO+ CLLCD a 1 DISP .10 WAIT
      IF a 2006 == THEN
        0 'a' STO
        "CENTENARIO DE LA FNI" MSGBOX
      END
    END
  »
»
```



Ejemplo 151. *Programa que calcula los factores primos de un número.*

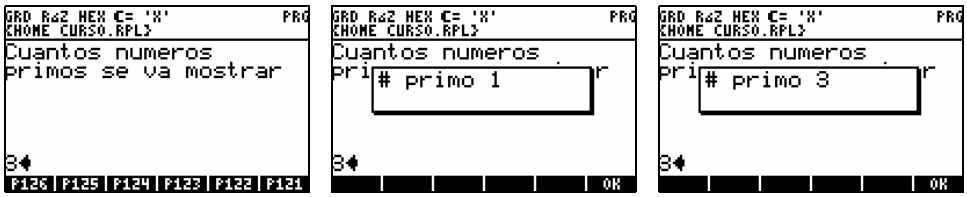
```
P125

« IFERR
  "Factores primos de
N=?"
  { "" { 1 0 } V }
  INPUT OBJ→ 2 → a b
  « {}
    WHILE a 1 >
    REPEAT
      IF a b MOD 0 == THEN
        b + a b / 'a' STO
      ELSE
        b 1 + 'b' STO
      END
    END
  »
  THEN DROP2 END
»
```



Ejemplo 152. *Programa que genera números primos.*

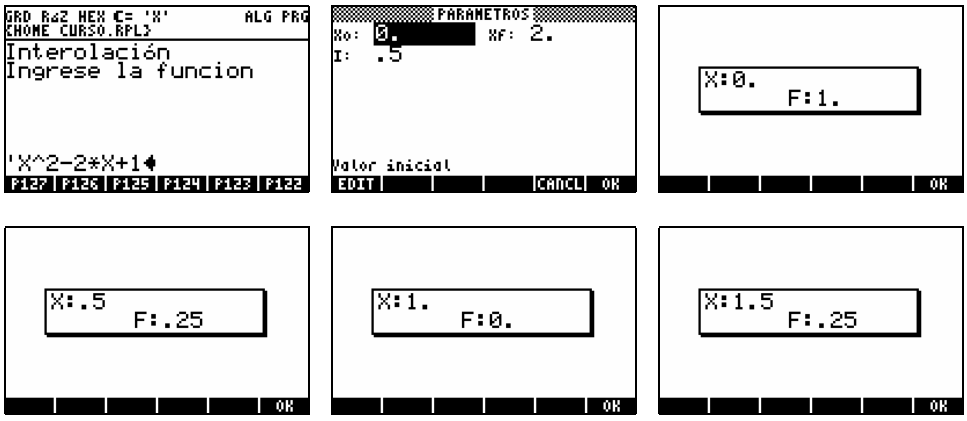
```
P126

« IFERR
  "Cuantos numeros
primos se va mostrar"
  { "" { 1 0 } V }
  INPUT OBJ→ DUP 1 SWAP
  FOR i
  i NEXT DUP →ARRY 0 → n a j
  « 2 n
    FOR i i 'j' STO
      WHILE i j * n ≤
      REPEAT
        0 'ⓘ(i*j)' STO
        'j' INCR DROP
      END
    NEXT
    1 n
```

```
    FOR i
      IF 'a(i)' 0 ≠ THEN
        "# primo " 'a(i)'
        EVAL + MSGBOX
      END
    NEXT
  »
  THEN
  DROP2 END
»
```



Ejemplo 153. *Programa que genera números primos.*

```
P127

« IFERR
  "Interolación
Ingrese la funcion"
  { "'" { 0 2 } ALG }
  INPUT OBJ→ → F
  « "PARAMETROS"
    { { "Xo:" "Valor inicial" 0 }
    { "Xf:" "Valor final" 0 }
    { "I:" "Intervalo" 0 } }
    { 2 3 } { } { } INFORM
    DROP EVAL → a b i
    « WHILE a b <
      REPEAT CLLCD
      "X:" a +  "
      F:" + a 'X' STO
      F →NUM + MSGBOX
      a i + 'a' STO
      END 'X' PURGE
    »
  »
  THEN
  DROP2 "Programa finalizado" DOERR
  END
»
```



Ejemplo 154. *Programa que crea el ingreso de datos de una matriz n x 4*

P128

```
« IFERR
  WHILE
  "Crear Matriz"
  { { "A" "Elemento 1 de 4" 0 }
  { "B" "Elemento 2 de 4" 0 }
  { "C" "Elemento 3 de 4" 0 }
  { "D" "elemento 4 de 4" 0 } }
  { 2 3 } { } DUP INFORM
  REPEAT
    IF DUP NOVAL POS THEN
      DROP CLLCD
      "Complete los datos" MSGBOX
    ELSE
      OBJ→ EVAL →ARRY
    END
  END DEPTH ROW→
  THEN CLEAR
  END
»
```