
***PROGRAMACIÓN EN USER-RPL
CON EJEMPLOS APLICADOS A
INGENIERÍA CIVIL***

***PROGRAMACIÓN EN USER-RPL
CON EJEMPLOS APLICADOS A
INGENIERÍA CIVIL***

***ANDRÉS R. GARCIA M.
INGENIERO CIVIL***

.....AQUI NO MOSTRABA PUSTULAS VISIBLES, SINO LA PUTREFACCION DE LAS
ALMAS MERCED AL ESTRAGO DEL MALIGNO SOBRE LAS CONCIENCIAS.

¡TUNJA, PUEBLO MALDITO!

PRESENTACIÓN

Con la aparición de las llamadas calculadoras de bolsillo y en especial de las calculadoras programables a finales de los años 80 se produce un cambio en la manera de ver y abordar los problemas convencionales de la Ingeniería y otras ciencias aplicadas dada la versatilidad y funcionalidad con que se desempeñan estas maquinas tanto en el ámbito académico como profesional.

La serie de calculadoras Hewlett Packard 48 y ahora la 49 ha demostrado ser una estas maquinas a las que hago referencia renglones atrás. Su funcionalidad e insuperable facilidad para crear aplicaciones a diferentes áreas de la ingeniería, la ha ubicado dentro de las calculadoras más populares dentro de nuestro medio. Pero esto no es todo, es una calculadora absolutamente programable hasta el límite de utilizarse en tareas como servir de maquina de video-juegos, emisor y receptor de infrarrojos para electrodomésticos, agenda personal y así mismo en el desarrollo y posterior utilización de excelentes aplicaciones a las diferentes asignaturas de la ingeniería, en este caso de la Ingeniería Civil como Estructuras, Hidráulica, Suelos, Vías y demás, permitiéndonos a los que estamos de una u otra forma ligados con esta; aprovechar al máximo estas fabulosas herramientas.

Refiriéndome ya a lo que es este libro, espero haber plasmado en las siguientes hojas, una manera sencilla de aprender a programar estas calculadoras, incluyendo diversos ejemplos desde el más sencillo hasta el mas elaborado para así lograr captar de manera cómoda y entendible lo que es la programación en **USER-RPL**, cuales son sus comandos y su correspondiente función y aplicación dentro de un programa. Cabe anotar que no todos los ejemplos propuestos en este libro son aplicados a Ingeniería civil, que todos los ejemplos están expuestos para HP 48G y HP 49G y que el lector de este libro sabe manejar su HP 48G O 49G muy bien y que además esta lo suficientemente familiarizado con la notación polaca inversa (RPN – REVERSE POLISH NOTATION)

Quiero agradecer a los autores de los programas que utilice para la elaboración de este libro como:

ROGER BRONCANO REYES por su gran aplicación HPUserEdit la cual permite escribir los programas en el computador en ambiente Windows. Este software es de libre distribución y lo pueden obtener en la siguiente dirección: www.hpcalc.org

SEBASTIEN CARLIER Y CHRISTOPH GIEBELINK Por el mejor emulador que haya visto: EMU48 1.10. Este software también es de libre distribución y lo pueden obtener también en www.hpcalc.org

ING. ANDRÉS R GARCÍA M
BOGOTA FEBRERO 12 DE 2006

CONTENIDO GENERAL

1. OBJETOS.

- 1.1 DEFINICIÓN.
- 1.2 TIPOS DE OBJETOS.
- 1.3 NÚMEROS DE IDENTIFICACIÓN DE LOS OBJETOS.

2. DELIMITADORES

- 2.1 DEFINICIÓN
- 2.2 TIPOS DE DELIMITADORES

3. VARIABLES

- 3.1 DEFINICIÓN
- 3.2 TIPOS DE VARIABLES

4. PROGRAMAS

- 4.1 DEFINICIÓN
- 4.2 INTRODUCCIÓN DE DATOS EN UN PROGRAMA.
- 4.3 OPERACIONES MATEMÁTICAS DENTRO DE UN PROGRAMA.
- 4.4 PROGRAMAS DE BLOQUE CONSTITUTIVO

5. PRESENTACIÓN DE DATOS Y RESULTADOS DE UN PROGRAMA

- 5.1 MEDIANTE EL COMANDO →TAG
- 5.2 MEDIANTE EL COMANDO DISP
- 5.3 MEDIANTE EL COMANDO MSGBOX
- 5.4 OTROS

6. MANIPULACIÓN DE LISTAS

- 6.1 DEFINICIÓN
- 6.2 COMO CREAR UNA LISTA
- 6.3 ADICIÓN DE OBJETOS A UNA LISTA
- 6.4 COMANDOS QUE PERMITEN OPERAR SOBRE LISTAS

7. ESTRUCTURA DE VARIABLE LOCAL

- 7.1 DEFINICIÓN
- 7.2 SINTAXIS DE VARIABLE LOCAL DENTRO DE UN PROGRAMA.
- 7.3 ESTRUCTURA DE VARIABLE LOCAL DENTRO DE UNA SUBRUTINA

8. FLAGS

8.1 DEFINICIÓN

8.2 COMO MODIFICAR LOS FLAGS DENTRO DE UN PROGRAMA

9. MENÚS

9.1 NÚMEROS DE ASIGNACIÓN DE LOS MENÚS

9.2 COMO CREAR MENÚS TEMPORALES

9.3 LOS MENÚS TEMPORALES DENTRO DE UN PROGRAMA.

10. ESTRUCTURAS DE PROGRAMACIÓN

10.1 DEFINICIÓN

10.2 ESTRUCTURAS CONDICIONALES

10.2.1 IF...THEN...END

10.2.2 IF...THEN...ELSE...END

10.2.3 CASE...THEN...END

10.2.3.1 ORGANIZACIÓN DEL TECLADO.

10.3 ESTRUCTURAS DE BUCLE

10.3.1 START...NEXT

10.3.2 FOR...NEXT

10.3.3 DO...UNTIL...END

10.3.4 WHILE...REPEAT...END

11. ESTRUCTURAS DE DETECCIÓN DE ERRORES

11.1 DEFINICIÓN

11.2 IFERR...THEN...END

12. VENTANA DE OPCIONES

12.1 DEFINICIÓN

12.2 EL COMANDO CHOOSE

12.3 FORMAS DE UTILIZACIÓN DEL COMANDO CHOOSE

13. PLANTILLAS DE ENTRADA

13.1 DEFINICIÓN

13.2 FORMAS DE UTILIZAR LAS PLANTILLAS DE ENTRADA

13.3 COMO PRESENTAR RESULTADOS A TRAVÉS DE UNA PLANTILLA.

14. EL ENTORNO SOLVR

14.1 COMO SOLUCIONAR ECUACIONES DENTRO DE UN PROGRAMA

15. EL ENTORNO PICT.

15.1 DEFINICIÓN

15.2 FORMAS DE VISUALIZAR EL ENTORNO PICT

15.3 COMO PONER Y VISUALIZAR TEXTO EN EL ENTORNO PICT.

15.4 COMO PONER Y VISUALIZAR GRÁFICOS EN EL ENTORNO PICT

15.5 COMANDOS QUE PERMITEN OPERAR SOBRE EL ENTORNO PICT

16 LIBRERÍAS

16.1 DEFINICIÓN

16.2 COMO INSTALAR Y DESINSTALAR LIBRERÍAS.

16.3 COMO CREAR LIBRERÍAS

BIBLIOGRAFIA



1. OBJETOS

1. OBJETOS

1.1 DEFINICIÓN.

Los objetos se definen como los elementos básicos de información utilizados por la HP. Por ejemplo, un número real, una ecuación, una lista, una secuencia o un programa son objetos.

1.2 TIPOS DE OBJETOS

Todos los objetos utilizados por la HP tienen un número de identificación. Así los podemos identificar fácilmente dentro de un programa.

Las siguientes tablas nos muestran el tipo de objeto, un ejemplo de dicho objeto y su respectivo número de identificación.

TIPO DE OBJETO	NUMERO
Numero real	0
Numero complejo	1
Secuencia de caracteres	2
Sistema real	3
Sistema complejo	4
Lista	5
Nombre global	6
Nombre local	7
Programa	8
Objeto algebraico	9
Entero binario	10
Objeto de gráficos	11
Objeto etiquetado	12
Objeto de unidades	13
Nombre de XLIB	14
Directorio	15
Biblioteca	16
Objeto de seguridad	17

Como se puede ver, todos los objetos tienen su número de asignación. También existen otros tipos de objetos que para efectos de los ejemplos propuestos en este libro no se tendrán en cuenta.

Por supuesto hay un comando que nos permite conocer los tipos de objetos en función de su número de identificación. Este comando se llama **TYPE**, y opera de la siguiente forma:

Ponemos el objeto, cualquiera que sea en el nivel 1 de la pila y ejecutamos el comando así:

Por ejemplo pongamos un número real en la pila

```
RAD XYZ HEX C~ 'X'  
(HOME) 10:48, MAY:21  
5:  
4:  
3:  
2:  
1: 235.  
ETI V% | | | | |
```

Ejecutamos el comando **TYPE** y obtendremos su numero de identificación en el nivel 1 de la pila así:

```
RAD XYZ HEX C~ 'X'
{HOME} 10 50, MAY:21
5:
4:
3:
2:
1: 0.
TYPE VTYPE PRG
```

EJEMPLO 2:

El mismo procedimiento pero ahora para un numero complejo:

```
RAD XYZ HEX C~ 'X' 10:53, MAY:21
{HOME}
5:
4:
3:
2:
1: (2.,6.)
TYPE VTYPE PRG
```

→

```
RAD XYZ HEX C~ 'X' 10 54, MAY:21
{HOME}
5:
4:
3:
2:
1: 1.
TYPE VTYPE PRG
```

Ponemos en la pila ejecutamos **TYPE** Obtenemos el numero de id
Mas adelante se explicara con ejemplos la importancia de conocer los números de identificación de los objetos

```
RAD XYZ HEX C~ 'X'  
{HOME} 10 58 MAY:21  
  
DELIMITADORES  
  
A | D | G | A | R | 12
```

2. DELIMITADORES.

2. DELIMITADORES.

2.1 DEFINICIÓN

Los delimitadores son "caracteres " que necesitan los objetos, para indicar de que tipo de objeto se trata. A continuación se presenta una tabla con los delimitadores más importantes y un ejemplo:

OBJETO	DELIMITADOR	EJEMPLO
Numero real	Ninguno	18.8
Numero complejo	()	(-6.5, 3.1)
Secuencias	" "	"Heavy Metal"
Sistemas	[]	[18 19 20]
Unidades	_	18_N
Programas	« »	« HOME CLVAR »
Operaciones algebraicas	' '	' X-Y'
Listas	{ }	{ 1 2 3 }
Comandos incorporados	Ninguno	RND
Nombres *	' '	'ARGM '

* Mas adelante veremos la conveniencia de llamar nombres sin estos delimitadores.

Así por ejemplo para introducir texto necesitamos delimitadores de secuencia que también llamaremos STRINGS.

Estos son los delimitadores que se usaran en los ejemplos presentados en este libro y son de importancia relevante en la utilización de objetos dentro de un programa como veremos mas adelante.

```
RAD XYZ HEX C~ 'X'  
(HOME) 11 07 MAY:21  
VARIABLES  
A N G A R 12
```

3. VARIABLES.

3. VARIABLES.

3.1 DEFINICIÓN

Una variable es cualquier objeto el cual esta almacenado bajo un nombre cualquiera, en el directorio HOME o en cualquier otro directorio. Para entenderlo mejor veamos el siguiente ejemplo: Supóngase que deseamos guardar nuestro nombre en el directorio actual bajo el nombre ES

PROCEDIMIENTO:

Lo primero que hacemos es poner nuestro nombre en la pila y luego el nombre bajo el cual lo deseamos almacenar así:

NOTA: Recordemos que para introducir texto debemos utilizar los delimitadores de **STRING**. " "

```
RAD XYZ HEX C~ 'X'
[HOME] 11:15, MAY:21
5:
4:
3:
2: "ANDRES GARCIA"
1: 'ES'
ASS DIS DISKY DOB DISN DB
```

Luego pulsamos la tecla **STO**:

```
RAD XYZ HEX C~ 'X'
[HOME] 11:16, MAY:21
5:
4:
3:
2:
1:
ES ETI VX
```

Variable creada

Observemos que en las teclas de menú ahora aparece ES. Para sacar a la pila el contenido de la variable ES simplemente pulsamos la tecla de menú ES y pondrá el contenido en la pila.

Así mismo como se almacenó nuestro nombre, podemos almacenar cualquier tipo de objetos como listas, Números reales, Programas etc. con solo poner el objeto en la pila, asignarle un nombre y grabarlo pulsando la tecla STO

```
RAD XYZ HEX C~ 'X'
[HOME] 11:19, MAY:21
5:
4:
3:
2:
1: "ANDRES GARCIA"
ES ETI VX
```

3.2 TIPOS DE VARIABLES

Existen dos tipos principales de variables como sigue

3.2.1 VARIABLES GLOBALES:

Las variables Globales son aquellas que se nombran en un programa principal y pueden ser reconocidas, en cualquier momento, desde cualquier bloque constitutivo de un programa.

Entiéndase por nombrar, el hecho de crear una variable de la manera como se hizo en el ejemplo de la pagina anterior.

Entiéndase por bloque constitutivo a las subrutinas que comprenden o que forman parte de un programa.

Estas dos definiciones anteriores se entenderán mucho mejor mas adelante cuando hablemos de lo que es un programa como tal.

3.2.2 VARIABLES LOCALES:

Las variables locales son variables provisionales creadas por un programa. Existen únicamente mientras se ejecuta el programa. Nunca aparecen en el menú VAR, es decir dentro de las teclas de menú del menú VAR. De igual manera este concepto será ampliado mas adelante cuando hablemos de la estructura de variable local en el numeral 7, pero es importante tener el concepto desde ahora.



4. PROGRAMAS.

4. PROGRAMAS.

Bien, después de haber aclarado algunos conceptos preliminares muy importantes hemos llegado a lo que realmente nos interesa ahora: Los programas.

4.1 DEFINICIÓN

Un programa en USER-RPL es un objeto definido por los delimitadores « » , que contienen una secuencia de números, comandos y otros objetos que se desean ejecutar de forma automática para realizar una tarea.

Pero en realidad los programas pueden contener algo mas que simples objetos, pueden asimismo contener estructuras como la estructura de variable local o la estructura de bifurcación de las que hablaremos mas adelante.

4.2 INTRODUCCIÓN DE DATOS EN UN PROGRAMA

La introducción de datos en un programa se puede hacer de varias maneras. Consideremos principalmente tres como sigue:

4.2.1 DIRECTAMENTE DESDE LA PILA

Para explicar esta manera de entrada de datos consideraremos el siguiente ejemplo:

Supóngase que deseamos calcular la suma de dos números cualquiera y dejar el resultado puesto en la pila.

PROCEDIMIENTO

Este procedimiento es muy sencillo. Como sigue:

```
RAD XYZ HEX C~ 'X'
[HOME] 11:29 MAY:21
5:
4:
3:
2: < + >
1: 'SUMA'
ASS DIS DISKY DOB DISN DE
```

En el nivel 2 tenemos el programa y en el nivel 1 el nombre que le vamos a asignar al programa.

Pulsamos STO

```
RAD XYZ HEX C~ 'X'
[HOME] 11:30 MAY:21
5:
4:
3:
2:
1:
SUMA ES ETI VX
```

Entonces lo que debemos hacer ahora es poner dos números en la pila y pulsar la tecla de menú **SUMA** la cual contiene el programa que suma dos números:

```
RAD XYZ HEX C~ 'X'
[HOME] 11:31, MAY:21
5:
4:
3:
2: 8.
1: 1.
SUMA ES ETI VX
```

Y ejecutamos la tecla de menú **SUMA**

```
RAD XYZ HEX C~ 'X'
[HOME] 11:32, MAY:21
5:
4:
3:
2:
1: 9.
SUMA ES ETI VX
```

Como se puede observar es un ejemplo muy sencillo que lo único que hace es sumar dos números que están puestos en la pila.

En este caso nuestra entrada de datos se hace directamente desde la pila. Ponemos los números previamente y ejecutamos el programa pulsando la tecla de menú **SUMA**.

4.2.2 MEDIANTE EL COMANDO INPUT

Esta es una manera mas elaborada de introducir datos dentro de un programa ya que la entrada esta dentro del programa como tal. El comando **INPUT** nos da la posibilidad que la introducción de datos se haga dentro del programa como tal y no tengamos que poner los datos en la pila y luego ejecutar el programa como en el ejemplo anterior.

El comando **INPUT** cuando se utiliza para entrada de datos, "casi" siempre va acompañado del comando **OBJ→** el cual nos quita los delimitadores de **STRING** del objeto entrado.

Para entender mejor esto vamos al siguiente ejemplo:

Vamos a considerar el mismo ejemplo anterior: sumar dos números

PROCEDIMIENTO:

Primero escribamos el programa sin el comando **OBJ→** para así entender mejor. La sintaxis del programa es la siguiente:

«	Abre programa	
"Primer numero" ""	El texto siempre va dentro de delimitadores de	string
INPUT	Nos permite hacer la entrada del primer dato	
"Segundo numero" ""		
INPUT	Nos permite hacer la entrada del segundo dato	
+	Suma los dos números	

»

Cierra programa

Así se vería en la pila:

```
RAD XYZ HEX C~ 'X'          11:38, MAY:21
[HOME]
2:
1: « "Primer numero"
   "" INPUT
   "Segundo numero" ""
   INPUT + »
SUMA | ES | ETI | VX |
```

Lo grabamos bajo el nombre SUMA Y luego lo ejecutamos: Nos aparecerá la siguiente pantalla al ejecutarlo:

Entonces introducimos cualquier numero

```
RAD XYZ HEX C~ 'X'          11:40, MAY:21
[HOME]
Primer numero
SUMA | ES | ETI | VX |
```

```
RAD XYZ HEX C~ 'X'          11:42, MAY:21
[HOME]
Primer numero
1998␣
SUMA | ES | ETI | VX |
```

Y luego pulsamos **ENTER** Seguirá lo siguiente

```
RAD XYZ HEX C~ 'X' PRG
[HOME] 11:43, MAY:21
Segundo numero
◀
SUMA ES ETI VX
```

Entonces introducimos cualquier numero

```
RAD XYZ HEX C~ 'X' PRG
[HOME] 11:44, MAY:21
Segundo numero
2004◀
SUMA ES ETI VX
```

Y luego pulsamos **ENTER**

Lo que sigue es que el programa procede a sumar los dos números. **Obsérvese el resultado**

```
RAD XYZ HEX C~ 'X' PRG
[HOME] 11:45, MAY:21
5:
4:
00:
2:
1: "19982004"
SUMA ES ETI VX
```

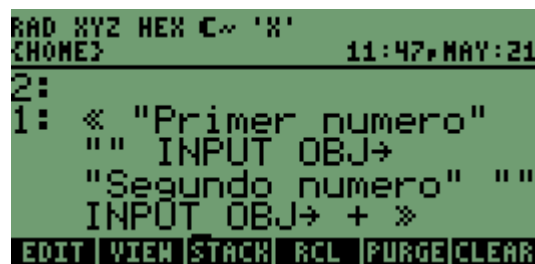
No es lo que esperábamos no es cierto?

Esto sucedió ya que inmediatamente después del **INPUT** no pusimos el comando que nos elimina los delimitadores de string entonces lo que hizo fue sumar dos string. Cuando sumamos dos strings la acción que se ejecuta es una concatenación de los dos strings que están puestos en la pila como se puede observar en la pantalla anterior.

Entonces escribámoslo con el comando que hizo falta:

```
«  
"Primer numero" ""  
INPUT OBJ→  
"Segundo numero" ""  
INPUT OBJ→  
+  
»
```

Así se vería en la pila:



```
RAD XYZ HEX C~ 'X'  
{HOME} 11:47, MAY: 21  
2:  
1: « "Primer numero"  
   "" INPUT OBJ→  
   "Segundo numero" ""  
   INPUT OBJ→ + »  
EDIT VIEW STACK RCL PURGE CLEAR
```

Lo grabamos bajo el nombre **SUMA** y lo ejecutamos:

Nos aparecerá lo mismo que para el ejemplo pero con una diferencia gigante:

LA RESPUESTA



```
RAD XYZ HEX C~ 'X'  
{HOME} 11:49, MAY: 21  
5:  
4:  
3:  
2:  
1: 4002.  
SUMA ES ETI VX
```

Espero que se haya entendido la función del comando **INPUT** y la función del comando **OBJ→** y la diferencia de poner solamente el **INPUT** sin el **OBJ→**

Estos dos son comandos muy sencillos pero de gran importancia y aplicación.

Al lector que tenga alguna experiencia previa en programación le parecerá innecesaria tanta explicación pero ese es el objetivo de este libro, hacer que las sintaxis de los programas se entiendan al máximo. Obviamente que a medida que avancemos se omitirán algunas explicaciones pero por ahora dejémoslo así.

Adicional a esto hay un comando de gran importancia dentro de la programación ya que nos permite recorrer paso a paso un programa, y considero necesario referenciarlo en este numeral. El comando se llama **DEBUG** y va acompañado de otro comando llamado **SST↓** y operan de la siguiente forma:

1. Vertimos el contenido del programa que queremos recorrer paso a paso en la pila. Esto se hace con la tecla de cambio derecha (la verde) y el nombre del programa que queremos recorrer paso a paso.

Primero pulsamos la tecla de cambio derecha y luego la tecla de menú donde este ubicado el programa a recorrer.

2. Pulsamos **DEBUG** y luego **SST↓** para recorrer el programa.

EJEMPLO:

Vamos a recorrer paso a paso el programa anterior, **SUMA**. Entonces seria así:

- Pulsamos la tecla de cambio derecha y luego la tecla de menú donde esta **SUMA** que en este caso es la primera tecla de menú. Saldrá la siguiente pantalla:

```
RAD XYZ HEX C~ 'X'
<HOME> 11:52, MAY:21
2:
1: « "Primer numero"
   "" INPUT OBJ→
   "Segundo numero" ""
   INPUT OBJ→ + »
SUMA ES ETI VX
```

Luego pulsamos **DEBUG**

```
RAD XYZ HEX C~ 'X' HLT
<HOME> 11:53, MAY:21
5:
4:
3:
2:
1:
DEBUG SST SST+ NEXT HALT KILL
```

Nos desaparece lo que esta en la pantalla y en el área de mensajes nos presenta el mensaje **HALT** lo que quiere decir que hemos interrumpido la ejecución de un programa. En este caso **SUMA**

Procedemos a pulsar **SST↓** hasta que finalice el programa. De esta forma nos daremos cuenta de que es lo que hace el programa **SUMA** paso a paso.

```
"Primer numero"
5:
4:
3:
2:
1: "Primer numero"
DEBUG SST SST+ NEXT HALT KILL
```



```
" "
5:
4:
3:
2: "Primer numero"
1: " "
DEBUG SST SST+ NEXT HALT KILL
```

```
RAD XYZ HEX C~ 'X' PRG
(HOME) 11:56, MAY:21
Primer numero
←
DEBUG SST SST+ NEXT HALT KILL
```



```
RAD XYZ HEX C~ 'X' PRG
(HOME) 11:56, MAY:21
Primer numero
1998
DEBUG SST SST+ NEXT HALT KILL
```

```
RAD XYZ HEX C~ 'X' HLT
(HOME) 11:57, MAY:21
5:
4:
3:
2:
1: "1998"
DEBUG SST SST+ NEXT HALT KILL
```



```
OBJ→
5:
4:
3:
2:
1: 1998.
DEBUG SST SST+ NEXT HALT KILL
```

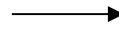
```
"Segundo numero"
5:
4:
3:
2: 1998.
1: "Segundo numero"
DEBUG SST SST+ NEXT HALT KILL
```



```
" "
5:
4:
3: 1998.
2: "Segundo numero"
1: " "
DEBUG SST SST+ NEXT HALT KILL
```

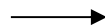


```
RAD XYZ HEX C~ 'X' PRG
(HOME) 11:59 MAY:21
Segundo numero
←
DEBUG SST SST+ NEXT HALT KILL
```



```
RAD XYZ HEX C~ 'X' PRG
(HOME) 12:00 MAY:22
Segundo numero
2004
DEBUG SST SST+ NEXT HALT KILL
```

```
RAD XYZ HEX C~ 'X' HLT
(HOME) 12:00 MAY:22
5:
4:
00:
2: 1998.
1: "2004"
DEBUG SST SST+ NEXT HALT KILL
```



```
OBJ→
5:
4:
00:
2: 1998.
1: 2004.
DEBUG SST SST+ NEXT HALT KILL
```

```
+
```

```
5:
4:
00:
2:
1: 4002.
DEBUG SST SST+ NEXT HALT KILL
```



```
⊗
```

```
5:
4:
00:
2:
1: 4002.
DEBUG SST SST+ NEXT HALT KILL
```

4.2.3 MEDIANTE UNA PLANTILLA DE ENTRADA

Esta es una forma bastante interesante de introducir datos dentro de un programa ya que muestra una presentación bastante amigable pero no se comentara todavía debido a que necesitamos conocer primero otros elementos de programación, y a que considero necesario dedicarle un numeral completo e esta forma de entrada de datos. Por lo tanto no se comentara si no hasta llegar al numeral 13.

4.3 OPERACIONES MATEMÁTICAS DENTRO DE UN PROGRAMA

Ya todos sabemos, la HP utiliza la notación polaca inversa (RPN) para sus operaciones, y funciones matemáticas que se ejecuten en la pila. Dentro de un programa podemos utilizar esta misma notación para operaciones matemáticas, pero además podemos escribir las operaciones tal cual se escriben en el papel.

Para que esto sea claro, vamos al siguiente ejemplo en el cual no-solo se explica lo anterior, además se utilizarán aplicaciones vistas anteriormente:

Escribamos un programa que nos calcule el área de un triangulo, la formula es:

$$A = (b * h) / 2$$

Donde:

A = Área

b = Base

h = Altura

Entonces escribiendo la operación en notación algebraica se vería así:

$$(b * h) / 2$$

Y escribiendo la operación en Notación Polaca Inversa se vería así:

$$b h * 2 /$$

Pero veamos el código escrito de ambas formas:

- UTILIZANDO LA OPERACIÓN EN MODO ALGEBRAICO:

```
« "Base" "" INPUT OBJ→
```

```
'b' STO
```

```
"Altura" "" INPUT OBJ→
```

```
'h' STO
```

```
'b*h/2' EVAL
```

```
»
```

Así se vería en la pila:

```

RAD XYZ DEC C~ 'X'
(HOME)
2:
1: « "Base" "" INPUT
   OBJ→ b STO "Altura"
   "" INPUT OBJ→ h STO
   'b*h/2.' EVAL »
OPER ABRIR DATOS INICI VX

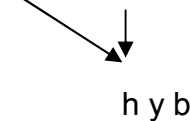
```

Grabamos bajo el nombre **OPER** y ejecutamos para $b = 10$ y $h = 15$. La respuesta se verá de la siguiente forma:

```

RAD XYZ DEC C~ 'X'
(HOME)
5:
4:
3:
2:
1:
75.
h b OPER ABRIR DATOS INICI

```



COMENTARIOS:

1. Obsérvese que se debe grabar el valor de la base bajo el nombre **b**, (**b STO**) y el valor de la altura bajo el nombre **h**, (**h STO**) de lo contrario no se hubiera realizado la operación. Tampoco se hubiera realizado la operación si hubiéramos grabado el valor de la base o de la altura con un nombre diferente a **b** o con **B** (mayúscula), ya que en la operación esta referenciado con **b**, y debe ser **b** (minúscula) ya que la HP discrimina entre mayúsculas y minúsculas.
2. De igual manera debemos percatarnos de que inmediatamente después de la operación debemos ejecutar el comando **EVAL** (evaluar) acción que no se ejecutaría si la operación estuviera en RPN
3. La operación siempre que este escrita en modo algebraico como en el anterior ejemplo, debe ir entre delimitadores de operación algebraica ' '
4. Al finalizar el programa nos dejo los valores de **b** y **h** grabados en el directorio actual, algo que para mi gusto es molesto. Mas adelante comentaremos la manera de evitar grabar valores para ejecutar una operación, esto mediante la creación de una variable local, o simplemente antes de finalizar el programa eliminar esta variable.
5. Importante también darnos cuenta de que para grabar el valor de **b** y **h** se utilizaron estos delimitadores ' '. Si no los hubiéramos utilizado para grabar **b** o **h** y existiera una variable grabada anteriormente con el mismo nombre se produciría un error y se cancelaría la ejecución del programa. De igual manera, si ya existiera una variable grabada anteriormente con el mismo nombre **b** o **h** pero utilizando los delimitadores ' ' para grabarla no se produciría error. Lo que

sucedería sería que la sobrescribiría es decir eliminaría el valor del antiguo b o h y lo cambiaría al valor de b o h que estamos introduciendo.

- Utilizando la operación en modo RPN

```
« "Base" "" INPUT OBJ→  
'b' STO  
"Altura" "" INPUT OBJ→  
'h' STO  
b h * 2 /  
»
```

Así se vería en la pila:



```
RAD XYZ DEC C~ 'X'  
{HOME}  
1: « "Base" "" INPUT  
OBJ→ 'b' STO  
"Altura" "" INPUT  
OBJ→ 'h' STO b h *  
2. / »  
h | b | OPER | ABRIR DATOS | INICI
```

COMENTARIOS:

1. No necesitamos el **EVAL** después de la operación

2. No necesitamos poner los delimitadores ' ' para la operación

Para ver la ejecución del programa paso a paso ejecute el comando **DBUG** anteriormente nombrado (se recomienda)

Estas son las formas básicas de hacer operaciones matemáticas proporcionadas por la HP. Personalmente en mi humilde opinión de programador, recomiendo ejecutar las operaciones en modo algebraico, ya que es mucho más fácil entenderlas si se les quiere hacer modificaciones posteriores.

A los usuarios de 49G les recomiendo utilizar su calculadora en modo RPN recordando que la 49 brinda la posibilidad de trabajar en modo algebraico también. Esto dejémoslo para otras calculadoras.

4.4 PROGRAMAS DE BLOQUE CONSTITUTIVO

Un programa de bloque constitutivo es un programa que se compone no-solo de una variable sino que se compone de varias variables que también se denominan subrutinas. Observemos el ejemplo anterior el cual para calcular el área de un triángulo solo necesito de una variable. Pero supongamos que el programa hubiera sido muy largo; es decir que hubiera tenido muchos más comandos para realizar otras tareas, esto es muy incómodo desde punto de vista de la edición del

programa: Editar variables muy largas es bastante incomodo y demorado. Entonces en ese caso la primera variable, es decir por donde empieza el programa; la referimos a segunda variable.

Hay ocasiones en que es obligatorio referirnos a una segunda variable o subrutina, es decir no solo lo hacemos por que la primera es una variable muy larga sino porque es totalmente necesario. También es importante aclarar que un programa puede estar compuesto de muchas subrutinas (hasta que la memoria de su HP se lo permita)

Ejemplo:

Supongamos que deseamos calcular el área y la longitud de circunferencia de un circulo dado su radio pero utilizaremos una variable para calcular el área y otra para calcular la longitud de circunferencia.

Como lo pueden imaginar estas son tareas que no representan programas muy largos. Se escribirá en dos variables simplemente para explicar como se crea una subrutina y como conectar las dos variables

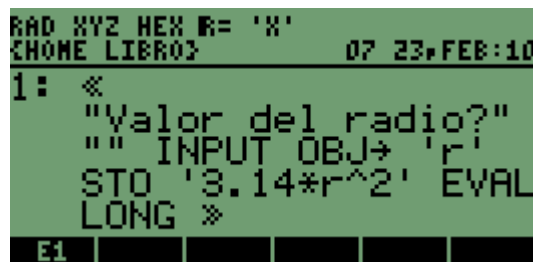
1. Creamos la variable por donde vamos a empezar el programa es decir la que va a calcular el área: La llamaremos AREA

2. La segunda variable, es decir; la que calculara la longitud de la circunferencia la llamaremos LONG

La operación matemática para calcular el área la escribiremos algebraicamente y entonces la sintaxis será la siguiente:

```
« "Valor del radio?" " "
INPUT OBJ→
'r'
STO
'3.14 * r^2 '
EVAL
LONG
»
```

En la pila se vería así:



The screenshot shows a calculator stack with the following content:

```
RAD XYZ HEX R= 'X'
CHOME LIBR03 07 23,FEB:10
1: «
"Valor del radio?"
" " INPUT OBJ→ 'r'
STO '3.14*r^2' EVAL
LONG »
E1
```

La grabamos bajo el nombre AREA: 'AREA' **STO**

COMENTARIOS:

1. Observemos que se grabo el valor del radio en el directorio actual bajo el nombre r 'r' **STO**

2. La operación es escrita algebraicamente por lo que tenemos que adicionar **EVAL** después de la operación

3. Esta es la observación más importante. Nótese que antes que se cierre el programa estamos llamando un nombre (LONG) Este es el nombre de la variable que sigue, es decir la que va a calcular la longitud de la circunferencia.

Aquí lo que se hizo fue que inmediatamente se acaban las tareas u operaciones por hacer en la variable ÁREA “nos vamos” para la variable LONG. Esta variable LONG debemos llamarla sin ninguna clase de delimitadores como se observa en la pantalla anterior.

Pero bueno, escribamos la variable LONG:

```
« '2 * 3.14 * r'
```

```
  EVAL
```

```
»
```

En la pila se vería así

```
RAD XYZ HEX R= 'X'
(HOME LIBRO) 07:28 FEB:10
5:
4:
3:
2:
1: « '2*3.14*r' EVAL »
OBJ+ →ARRY →LIST →STR →TAG →UNIT
```

Lo grabamos con el nombre LONG: 'LONG' STO

COMENTARIOS:

1. Percatarnos del EVAL después de la operación

Entonces en este momento debemos tener dos variables grabadas que son: ÁREA Y LONG

```
RAD XYZ HEX R= 'X'
(HOME LIBRO) 07:30 FEB:10
5:
4:
3:
2:
1:
LONG AREA E1
```



Variables

Entonces ejecutemos nuestro primer programa de bloque constitutivo. Recordemos que el programa inicia con la variable **ÁREA**
 Entonces para un radio de 8 obtendremos los siguientes valores:

```

RAD XYZ HEX R= 'X'          PRG
CHOME LIBRO3              07:36,FEB:10
Valor del radio?
8
LONG  AREA  E1

```

```

RAD XYZ HEX R= 'X'          PRG
CHOME LIBRO3              07:36,FEB:10
5:
4:
3:
2:                200.96
1:                50.24
r  LONG  AREA  E1

```

COMENTARIOS:

1. Este ejemplo es simplemente para explicar como se llama una subrutina dentro de un programa, ya que este programa en esencia; no necesitaba una subrutina dado que es muy corto

2. Fijémonos que el programa nos deja la variable r grabada en el directorio actual. Veamos como eliminar esta variable:

Lo único que tenemos que hacer es lo siguiente:

Antes de cerrar la subrutina debemos agregar lo siguiente

{r} PURGE que lo que hace es sacar el nombre a borrar a la pila en este caso r

y luego borrarlo mediante el comando **PURGE**

Esta ultima parte (borrar r) debemos ponerla al final de la subrutina (en este caso LONG) y no al final del programa principal (en este caso AREA) ya que si lo ponemos al final del programa principal AREA, cuando nos vayamos a la subrutina LONG no calculara nada porque el valor de r ya no existe, se ha borrado

Entonces el código completo quedara así

AREA:

```

« "Valor del radio?" " "
INPUT OBJ→ 'r' STO
' 3.14 * r^2 ' EVAL
LONG »
LONG:
« ' 2 * 3.14 * r ' EVAL {r} PURGE
»

```

Agregando este ultimo comando ya no nos quedara la variable r en el directorio actual.

Observemos que la variable a borrar debe ir entre delimitadores de lista.

Para terminar con este numeral espero que haya quedado bien clara la manera como se llaman las subrutinas dentro de un programa. **Simplemente se llaman por su nombre y sin ningún delimitador.**

Recomiendo que le pasen el **DEBUG** (comando nombrado anteriormente) a este pequeño programa que acabamos de hacer (ÁREA) para así lograr una comprensión total de lo que es un programa de bloque constitutivo.

```
RAD XYZ HEX R= 'X'  
[HOME LIBRO] 07 41 FEB:10  
PRESENTACION DE  
RESULTADOS  
A N G A R 18
```

5. PRESENTACIÓN DE RESULTADOS.

5. PRESENTACIÓN DE RESULTADOS.

La presentación de los resultados que arroja un programa es de vital importancia ya que de esto depende que los potenciales usuarios de nuestras creaciones entiendan e interpreten lo que sale a la pila después de ejecutar o correr un programa. Notemos que los programas que hemos escrito hasta el momento nos arrojan o nos ponen resultados en la pila pero no podemos interpretarlos de manera correcta dado que solo nos pone números en la pila pero no podemos saber ni interpretar que representan.

Para la presentación de resultados existen varios comandos, unos más funcionales que otros pero todos igualmente importantes y de fácil manipulación.

A continuación se presentan comandos que nos permiten "etiquetar" los resultados o presentarlos con su respectivo nombre, así por ejemplo un programa que nos calcule el volumen de una esfera dado el radio, nos presentara el resultado con su respectivo nombre: V = 35 por ejemplo. En este numeral haremos referencias a solo algunos de estos comandos, los demás los referiremos mas adelante.

5.1 MEDIANTE EL COMANDO →TAG

Este comando es el mas sencillo para etiquetar los resultados salidos de un programa. Para ver como funciona consideremos el siguiente ejemplo sencillo:

EJEMPLO 1:

Escribamos un programa que nos calcule las soluciones de una ecuación cuadrática.

Recordemos la forma de la ecuación cuadrática:

$$a * x^2 + b * x + c = 0$$

Y la formula para hallar las soluciones serán:

$$X1 = (-b + \sqrt{b^2 - 4 * a * c}) / 2 * a$$

$$X2 = (-b - \sqrt{b^2 - 4 * a * c}) / 2 * a$$

Esto Suponiendo que la variable sea X

Bueno, se supone que ustedes manejan las matemáticas perfectamente por lo que de ahora en adelante no daré tantos detalles.

Para el ejemplo los valores de los coeficientes serán a= 1 , b = 2 , c = 1

Vamos al código:

```
«  
"Valor de a?" ""  
INPUT OBJ→  
'a' STO  
"Valor de b?" " "  
INPUT OBJ→  
'b' STO  
"Valor de c?" " "  
INPUT OBJ→  
'c' STO  
'(-b+√(b^2-4*a*c))/(2*a)' EVAL X1 →TAG  
'(-b-√(b^2-4*a*c))/(2*a)' EVAL X2 →TAG  
»
```

Lo grabamos bajo el nombre CUAD y ejecutamos:

Tendrá que aparecernos algo como lo que sigue:

```

RAD XYZ HEX R= 'X'          PRG
[HOME LIBR03]             07:53,FEB:10
Valor de a?

1♦
CUAD  E1

```

```

RAD XYZ HEX R= 'X'          PRG
[HOME LIBR03]             07:54,FEB:10
Valor de b?

2♦
a  CUAD  E1

```

COMENTARIOS:

1. Para comenzar observemos que el programa nos grabo los valores de a, b , c en el directorio actual. Esto se evita poniendo al final del programa { a b c} **PURGE** (pruébenlo como ejercicio) En los ejemplos que hagamos de aquí en adelante se incluirá esta ultima parte para borrar la variables grabadas

```

RAD XYZ HEX R= 'X'          PRG
[HOME LIBR03]             07:55,FEB:10
Valor de c?

1♦
b  a  CUAD  E1

```

```

RAD XYZ HEX R= 'X'          PRG
[HOME LIBR03]             07:56,FEB:10
5:
4:
3:
2:          X1: -1
1:          X2: -1
c  b  a  CUAD  E1

```

2. Notemos que inmediatamente después del **EVAL** aparece X. **Este texto es el que va a etiquetar el resultado.** Inmediatamente después aparece el comando que genera la etiqueta: **→TAG**
3. Importantísimo pasarle el **DEBUG** a este programa para que vean lo que hace paso a paso.
4. Para que el resultado se vea de esta manera debemos modificar primero el siguiente **FLAG** :

```

SYSTEM FLAGS
76 Purge confirm      +
✓79 Std stack
80 EQM cur stk font
81 GRB Alg cur font
82 EQM edit cur font
83 Display grobs on
85 Normal stk disp   +
      ✓CHK  CANCL  OK

```

Es decir dejarlo como **Std stack** que significa pila estándar. De lo contrario se vería así:

```

RAD XYZ HEX R= 'X'
CHOME LIBRO3 08 02 FEB:10
5:
4:
3:
2: X1:(-1)
1: X2:(-1)
c b a CUAD E1

```

Esto seria desactivando el FLAG:

```

SYSTEM FLAGS
76 Purge confirm +
79 Algebraic stk
80 EQM cur stk font
81 GRB Alg cur font
82 EQM edit cur font
83 Display grobs on
85 Normal stk disp +
CHK CANCL OK

```

Mas adelante veremos como se modifican los FLAGS especificamente dentro del programa. Realicémoslo de nuevo pero cambiemos la etiqueta a SOLUCION1 y SOLUCION2 y adiccionémosle la parte final que borra las variables grabadas. Utilicemos los mismos valores que para el ejemplo anterior (1 2 1)

```

«
"Valor de a?" ""
INPUT OBJ→
'a' STO
"Valor de b?" " "
INPUT OBJ→
'b' STO
"Valor de c?" " "
INPUT OBJ→
'c' STO
'(-b+√(b^2-4*a*c))/(2*a)' EVAL SOLUCION1 →TAG
'(-b-√(b^2-4*a*c))/(2*a)' EVAL SOLUCION2 →TAG
{ a b c } PURGE
»

```

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 08 09,FEB:10
5:
4:
3:
2: SOLUCION1: -1
1: SOLUCION2: -1
CUAD E1

```

COMENTARIOS:

1. Notemos que borramos los valores que habíamos grabado
2. Cambiamos las etiquetas a SOLUCION1 y SOLUCION2
3. Fijémonos que el nombre SOLUCIÓN quedo pegado el 1 es decir quedo SOLUCION1. Para que queden separados debemos poner el texto de la etiqueta entre delimitadores de STRING. Como sigue:

```

«
"Valor de a?" ""
INPUT OBJ→
'a' STO
"Valor de b?" " "
INPUT OBJ→
'b' STO
"Valor de c?" " "
INPUT OBJ→
'c' STO
'(-b+√(b^2-4*a*c))/(2*a)' EVAL "SOLUCION 1" →TAG
'(-b-√(b^2-4*a*c))/(2*a)' EVAL "SOLUCION 2" →TAG
{ a b c } PURGE
»

```

Al ejecutar este código, es decir; al correr este programa la respuestas quedarían presentadas de la siguiente forma:

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 08 11,FEB:10
5:
4:
3:
2: SOLUCION 1: -1
1: SOLUCION 2: -1
CUAD E1

```

COMENTARIOS:

1. Observemos que ahora si quedaron despegados SOLUCIÓN y 1. No olvidemos que para poner texto que contenga espacios y que vaya a etiquetar algún resultado, debemos ponerlo con delimitadores de STRING

Espero se haya entendido como se presentan resultados mediante el comando →TAG, es bastante fácil.

5.2 MEDIANTE EL COMANDO MSGBOX

Este es otro comando bastante funcional ya que nos permite mostrar resultados pero ahora en una ventana de dialogo. Consideremos el siguiente ejemplo para aplicar el comando:

Escribamos un programa (muy sencillo) para calcular las perdidas por fricción en una tubería simple utilizando la Ecuación de Darcy:

$$hf = f * (L/D) * (V^2/2*g) \quad \text{Donde:}$$

f = Factor de fricción

L = Longitud de la tubería

D = Diámetro. (Usualmente se designa con la letra griega ϕ)

V = Velocidad de flujo

g = Aceleración de la gravedad (9.81 m / s²)

Para el ejemplo utilizaremos los siguientes valores:

$$f = 0.012$$

$$L = 200 \text{ m}$$

$$D = 0.254 \text{ m}$$

$$V = 3.94 \text{ m / s}$$

$$g = 9.81 \text{ m / s}^2$$

Vamos al código:

```
« "Factor de fricción?" " "
INPUT OBJ→ 'f' STO
"Longitud de la
tubería (En m)" " "
INPUT OBJ→ 'L' STO
"Diametro de la
tubería? (EN m)" " "
INPUT OBJ→ 'D' STO
"Velocidad de flujo?
(En m/s)" " "
INPUT OBJ→ 'V' STO
' f * (L/D) * V^2/19.62 ' EVAL
→STR "hf= " SWAP + MSGBOX
{ f L D V } PURGE
»
```

Así vería parte del código en la pila:

```

RAD XYZ HEX R= 'X'
CHOME LIBRO3 08:21,FEB:10
1: «
  "Factor de fricció...
  "" INPUT OBJ→ 'f'
  STO
  "Longitud de la=t...
CUAD E1

```

Grabamos bajo el nombre PERD ('PERD' STO) y lo ejecutamos:

```

RAD XYZ HEX R= 'X' PRG
CHOME LIBRO3 08:24,FEB:10
Factor de fricción?
0.012
PERD CUAD E1

```

```

RAD XYZ HEX R= 'X' PRG
CHOME LIBRO3 08:25,FEB:10
Longitud de la=
tubería (En m)
200
F PERD CUAD E1

```

```

RAD XYZ HEX R= 'X' PRG
CHOME LIBRO3 08:26,FEB:10
Diametro de la=
tubería? (EN m)
0.254
L F PERD CUAD E1

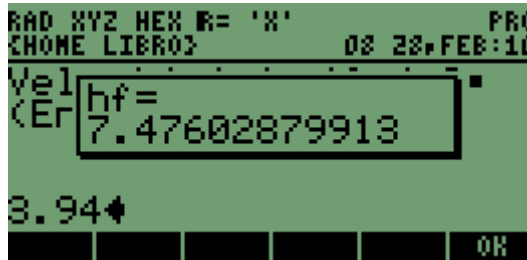
```

```

RAD XYZ HEX R= 'X' PRG
CHOME LIBRO3 08:26,FEB:10
Velocidad de flujo=
(En m/s)
3.94
D L F PERD CUAD E1

```

La respuesta se vera de la siguiente forma:



COMENTARIOS:

1. Observemos en las pantallas anteriores como van apareciendo las variables que vamos grabando.

2. Notemos el **EVAL** inmediatamente después de la operación y después viene lo más interesante.

3. Después del **EVAL** aparece un comando nuevo para nosotros: **→STR**

Este comando lo que hace es convertir el resultado de la operación en un STRING, es decir le pone delimitadores de secuencia ya que para poder mostrarlo aplicando el comando **MSGBOX** tiene que estar dentro de delimitadores de secuencia o STRING

Luego viene el texto hf el cual es la etiqueta que le vamos a poner al resultado. Observemos que este texto esta dentro de delimitadores de STRING

Después de esto viene otro comando nuevo: **SWAP** . Este comando lo que hace es voltear los dos datos que hay en la pila en ese momento que son "7.4798..." y "hf=".

¿Pero porque se voltean?

Simplemente porque si sumamos estos dos STRINGS (nótese que a continuación viene un +) tal cual nos los arrojó el programa la respuesta nos hubiera quedado al revés. Es decir así:

A continuación viene un + que es el que suma los dos STRINGS que son:

3.1 La respuesta arrojada por la operación "7.4798..."

3.2 La etiqueta para la respuesta " hf = "

Para que todo esto se entienda mejor háganle un **DEBUG** a este programa y así entenderán perfectamente.

Luego viene el comando **MSGBOX** que lo que hace es coger el STRING que se encuentre en la pila y presentarlo en una ventana. En este caso el STRING que estaba en la pila era " hf = 7.4798....."

A continuación viene **{ f L D V } PURGE** que borra las variables que habíamos grabado: **{ f L D V }**

4. Fijémonos que al presentar el resultado, nos queda de fondo la pantalla con el ultimo **INPUT** .

Algo que en mi concepto es antiestético a menos que fuese totalmente necesario. Para solucionar este problema lo que debemos hacer es poner el comando **CLLCD** antes del comando **MSGBOX** .

Este comando **CLLCD** lo que hace es borrar la pantalla pero no borra la pila en si, es decir; no borra los datos que hay en la pila y para estos casos siempre va antes del comando **MSGBOX**

Veamos como quedaría el código con este nuevo comando y como se vería la respuesta:

« "Factor de fricción?" " "

INPUT OBJ→ 'f' STO

"Longitud de la
tubería (En m)" " "

INPUT OBJ→ 'L' STO

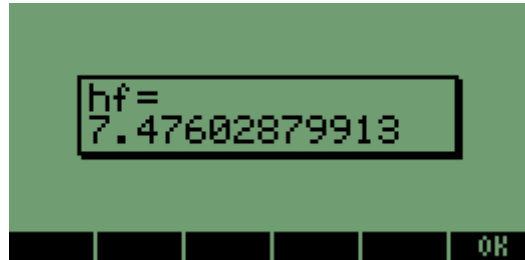
"Diametro de la

```

tuberia? (EN m)" " "
INPUT OBJ→ 'D' STO
"Velocidad de flujo?
(En m/s)" " "
INPUT OBJ→ 'V' STO
' f * (L/D) * V^2/19.62 ' EVAL
→STR "hf= " SWAP + CLLCD MSGBOX
{ f L D V } PURGE
»

```

Lo grabamos bajo el mismo nombre PERD y lo ejecutamos. Entonces la respuesta se vería así:



Mucho mejor no les parece?

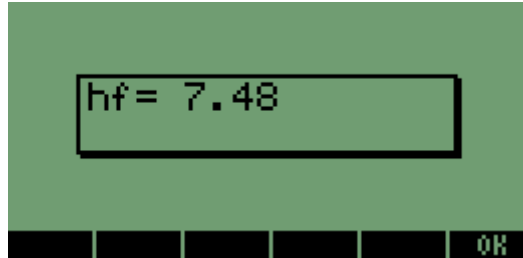
5. Supongamos que no queremos ver la respuesta con tantos lugares decimales. Entonces lo que debemos hacer es poner el comando **RND** el cual funciona de la siguiente manera: Inmediatamente después del número o respuesta que queremos "cortar" le ponemos el número de lugares decimales que queremos y luego el comando **RND**. Pero veamos como quedaría el código del ejemplo anterior con el nuevo comando **RND** Supondremos 2 espacios decimales:

```

« "Factor de fricción?" " " "
INPUT OBJ→ 'f' STO
"Longitud de la
tubería (En m)" " "
INPUT OBJ→ 'L' STO
"Diametro de la
tuberia? (EN m)" " "
INPUT OBJ→ 'D' STO
"Velocidad de flujo?
(En m/s)" " "
INPUT OBJ→ 'V' STO
' f * (L/D) * V^2/19.62 ' EVAL 2 RND
→STR "hf= " SWAP + CLLCD MSGBOX
{ f L D V } PURGE
»

```

Veamos como queda la respuesta:



Vemos que “recorto” la respuesta a dos lugares decimales.
 De nuevo recomiendo hacerle un **DEBUG** a este código
 Espero haber sido lo suficientemente claro y explícito al explicar estos nuevos comandos que acabamos de ver (→**STR**, **MSGBOX**, **CLLCD**, **SWAP**, **RND**)

5.3 MEDIANTE EL COMANDO DISP

La presentación de datos o resultados mediante este comando es en mi concepto, un poco mas funcional que los otras dos comandos vistos anteriormente (→TAG y MSGBOX) ya que nos permite utilizar toda el área de la pantalla para tal fin. El DISP es un comando que suele ir acompañado de otros comandos, a los cuales también haremos referencia en este numeral.

Este comando no solamente nos sirve para presentar resultados arrojados de determinada operación si no que además nos permite poner texto en la pantalla de manera organizada y además los códigos ocupan poco espacio en memoria.

Primero consideraremos un ejemplo en donde lo que se va a presentar es una respuesta arrojada por un pequeño programa y luego mediante el mismo comando DISP haremos una sencilla presentación para el mismo programa. Esto evocando la posibilidad que nos ofrece DISP de poner texto en pantalla.

Vamos entonces al primer ejemplo:

Vamos a considerar el mismo ejemplo anterior en donde calculamos las perdidas por fricción en una conducción simple con los mismos valores pero además vamos a calcular el área de la sección de la tubería:

Veamos entonces al código incluyendo el nuevo comando DISP y sus comandos “acompañantes”

```
« "Factor de fricción?" ""
INPUT OBJ→ ' f ' STO
"Longitud de la
tubería (En m)" "" INPUT OBJ→
' L ' STO
"Diametro de la
tubería? (EN m)" "" INPUT OBJ→
' D ' STO
"Velocidad de flujo?
(En m/s)" "" INPUT OBJ→ ' V ' STO
'f*(L/D)*(V^2/19.61)' EVAL 2
RND →STR "hf= " SWAP + CLLCD
1 DISP '3.14*(D/2)^2' EVAL 2
RND →STR "Area= " SWAP + 2
DISP 3 FREEZE { V D L f }
PURGE
```

»

Lo grabamos bajo el nombre PERD2 y ejecutamos
Entonces ahora nos va a arrojar dos respuestas que son: Perdidas y Área de la sección de la tubería los cuales se ven así:

```
hf= 7.48  
Area= .05  
  
PERD CUAD E1
```

COMENTARIOS:

1. Notemos que después de haber concatenado los dos STRINGS con el + (→STR "hf= " SWAP +) aparece el comando **1 DISP** donde: **1** es el nivel de la pila donde vamos a ubicar el resultado pero ordenado de arriba abajo es decir el nivel 1 ya no es el primero de abajo hacia arriba sino de arriba abajo.

2. Luego aparece **DISP** el cual nos permite poner el resultado en el nivel especificado en este caso 1.

Sucede lo mismo para el segundo resultado, después de haber concatenado los dos STRINGS con el + (→STR "Area= " SWAP + 2 DISP) aparece el comando **2 DISP** donde: **2** es el nivel de la pila donde vamos a ubicar el resultado.

Luego aparece **DISP** el cual nos permite poner el resultado en el nivel especificado en este caso 2. **Para usuarios de 48G es importante aclararles que disponen de 7 niveles para poner resultados en pantalla.**

Luego viene un comando nuevo: **FREEZE**

FREEZE nos permite congelar la pantalla para que el resultado se mantenga visible, sin el **FREEZE** el resultado se hubiera puesto si en los niveles especificados pero hubiera desaparecido tan rápido que ni nos hubiéramos dado cuenta.

Fijémonos que el **FREEZE** viene acompañado de un numero en este caso 3. Este numero indica que el área de la pantalla a congelar. En este caso consideramos 3 ya que es un valor que se ajusta perfectamente a cualquier ejemplo.

3. Por ultimo aparece { V D L f } **PURGE** que es el encargado de borrar las variables que habíamos grabado.

Existe otro comando que nos permite mantener los resultados visibles en la pantalla. Este comando se denomina **WAIT**.

El comando **WAIT** lo que hace es que después de poner nuestros resultados en los niveles deseados nos espere un tiempo (en segundos) determinado.

Si ponemos **5 WAIT** nos dejara visibles los resultados durante 5 segundos, si ponemos **10 WAIT** nos dejara visibles los resultados durante 10 segundos y si ponemos **0 WAIT** nos dejara visibles los resultados **hasta que se pulse una tecla** que es mucho mas conveniente en este caso.

Pero veamos nuestro ejemplo ahora utilizando el **WAIT**

```

« "Factor de friccion?" ""
INPUT OBJ→ ' f ' STO
"Longitud de la
tuberia (En m)" "" INPUT OBJ→
' L ' STO
"Diametro de la
tuberia? (EN m)" "" INPUT OBJ→
' D ' STO
"Velocidad de flujo?
(En m/s)" "" INPUT OBJ→ ' V ' STO
'f*(L/D)*(V^2/19.61)' EVAL 2
RND →STR "hf= " SWAP + CLLCD
1 DISP '3.14*(D/2)^2' EVAL 2
RND →STR "Area= " SWAP + 2
DISP 0 WAIT {V D L f}
PURGE

```

»

Grabamos bajo el mismo nombre PERD2 y ejecutamos. Se vera de la siguiente forma:

COMENTARIOS:

1. Observemos que nos visualiza el reloj. Esto porque la opción de ver reloj estaba activada. Mas adelante veremos como modificar esto (**FLAGS**) para que al presentar respuestas no nos deje visible el reloj. Por ahora dejémoslo así.

2. Si presionamos **ENTER** o cualquier otra tecla nos dejara un numero puesto en el nivel 1 de la pila. Mas adelante veremos que significa este numero (**Organización del teclado**)

3. Fijémonos que el **0 WAIT** va después de que los resultados han sido puestos en sus respectivos niveles. Si lo ponemos antes pues no tiene ningún sentido ya que no hay ningún resultado que mostrar.

Si ubicamos el **0 WAIT** después de haber puesto únicamente el primer resultado pues no nos mostrara el segundo. El **0 WAIT** debe ubicarse una vez estén ubicados los resultados o los datos que queremos mostrar en sus respectivos niveles.

Vamos entonces a la segunda parte del ejemplo: Crear una pequeña presentación para el programa PERD2 mediante la utilización del comando **DISP**. Desactivemos la opción de mostrar el reloj para que no nos dañe nuestra presentación.

Vamos a suponer que los datos que lleva nuestra presentación son los siguientes:

Nombre del programa
Nombre del creador del programa
Versión
Año.

Y por ultimo una indicación para continuar el programa. En este caso le vamos a poner **PRESIONE ENTER**

Veamos el código:

```
« CLLCD
"   PERDIDAS

POR: Joe Satriani
  VERSION 1.0
    2005
PRESIONE ENTER "
1 DISP 0 WAIT
»
```

Grabamos bajo el nombre PRES y ejecutamos. Se vera algo como esto.



COMENTARIOS:

1. Notemos que primero va el comando **CLLCD** para que nos ponga la pantalla en blanco, es decir nos borre la pantalla.
2. El texto va dentro de delimitadores de secuencia o STRING con sus respectivos espacios para que quede centrado en la pantalla.
3. Observemos que necesitamos un solo **DISP** para poner todo el texto en pantalla y no uno para cada línea de texto. Esto debido a que todo el texto lo pusimos dentro de **un solo delimitador de STRING**.
4. El texto esta colocado en el nivel 1 (**1 DISP**) pero lo mostró en varios niveles de la pantalla. Esto debido a que le dimos espacios entre líneas para que quedara centrado y bien presentado

Ahora la idea principal es concatenar las dos variables que constituyen nuestro programa que son PERD2 Y PRES para que una vez presionamos ENTER después de la presentación se ejecute la variable PERD2.

Esta tarea la dejo para que el lector la haga ya que es bastante fácil y además se explico unas paginas atrás.

Espero que hayan quedado suficientemente claros los comandos que nos permiten visualizar datos y resultados en pantalla, ya que de aquí en adelante los utilizaremos bastante en los demás ejemplos propuestos.

5.4 OTROS

Existen otros comandos que también nos permiten visualizar resultados o cualquier otro objeto pero ahora utilizando códigos mucho más elaborados y haciendo uso del entorno de gráficos **PICT**. Estos comandos los analizaremos mas adelante ya que para entenderlos y manejarlos correctamente necesitamos algunas explicaciones previas además de otros elementos de programación.

```
RAD XYZ HEX R= 'X'  
<HOME LIBR0> 09 12 FEB 10  
MANIPULACION DE  
LISTAS  
A | D | G | A | R | 12
```

6. MANIPULACIÓN DE LISTAS.

6. MANIPULACIÓN DE LISTAS.

6.1 DEFINICIÓN

Para la HP una lista es todo lo que este dentro de los delimitadores { }.

El buen manejo de listas es quizás uno de los elementos de programación más importante que hay, dado que las listas son muy flexibles para manejar y operar sobre datos dentro de un programa.

Unos ejemplos de lista pueden ser los siguientes:

- { A B C D }

- { 1 2 3 4 }

- { "TRASH METAL" "MI METAL-TENDENCIA FAVORITA" }

Las listas pueden contener cualquier tipo de objeto (números reales, números complejos secuencias o STRINGS, unidades, programas, otras listas etc.)

6.2 COMO CREAR UNA LISTA

Primero que todo vamos a ver como se crea una lista directamente desde la pila y luego veremos como se crea aplicando un programa.

6.2.1 DESDE LA PILA

Para crear una lista debemos tener puestos en la pila los diferentes elementos que van a conformar la lista, especificar el numero de elementos que la conformaran y luego si ejecutar el comando →LIST

Veamos entonces el siguiente ejemplo:

Supongamos que los elementos que conformaran la lista son los siguientes:

1 2 3 4

Así los veríamos en la pila:

```
RAD XYZ HEX R= 'X'
[HOME LIBRO3] 09:17,FEB:10
5:
4:
3:
2:
1:
PRES E1
```

Ahora debemos especificar el numero de elementos que conformaran la lista. En este caso son cuatro elementos:

Entonces ejecutamos el comando →LIST

```
RAD XYZ HEX R= 'X'
[HOME LIBRO3] 09:20,FEB:10
5:
4:
3:
2:
1: ( 1 2 3 4 )
ELEN PROC OBJ+ →LIST SUB REPL
```

Pero bueno, aquí sabíamos cuantos elementos conformarían la lista, pero consideremos un caso en que no lo supiéramos.

Para esto hay otro comando muy interesante que nos indica cuantos objetos (y digo objetos porque puede tratarse no solo de números reales) hay en la pila.

Veamos otro ejemplo aplicando este comando:

Los objetos que conformarían la lista son los siguientes:

(2,1) (1,2) (3,3) (5,2)

Así los veríamos en la pila:

```
RAD XYZ HEX R= 'X'
<HOME LIBR03> 09:24,FEB:10
5:
4: (2.,1.)
3: (1.,2.)
2: (3.,3.)
1: (5.,2.)
ELEM PROC OBJ+ →LIST SUB REFL
```

Entonces ejecutamos el comando **DEPTH** y luego el comando **→LIST** y veremos algo así:

```
RAD XYZ HEX R= 'X'
<HOME LIBR03> 09:25,FEB:10
4:
3:
2:
1: ( (2.,1.) (1.,2.)
   (3.,3.) (5.,2.) )
ELEM PROC OBJ+ →LIST SUB REFL
```

De esta manera se crean las listas directamente desde la pila, pero ahora que tenemos el concepto veamos como se hace con un programa.

6.2.1 MEDIANTE UN PROGRAMA

Para explicar esta manera de crear listas vamos a considerar el siguiente ejemplo:

Escribamos un programa que nos pida una entrada de datos pero estos datos ya no iran separados; es decir ya no haremos un INPUT para cada dato sino que pediremos la entrada de los datos uno tras otro

Entonces consideremos una serie de datos de diámetros de tuberías en pulgadas y los convertiremos a metros. Los datos son los siguientes:

6 8 10 12 18 20 24 28 32 (Todos están pulgadas)

Veamos entonces el código:

```
« "Diametros en
pulgadas" " " INPUT OBJ→
DEPTH →LIST 0.0254 *
»
```

Así lo vemos en la pila:

```
RAD XYZ HEX R= 'X'
<HOME LIBR03> 09:33,FEB:10
1: «
   "Diametros en=#pul...
   " " INPUT OBJ→
   DEPTH →LIST .0254 *
   »
OBJ+ →ARRY →LIST →STR →TAG →UNIT
```


Grabamos bajo el nombre CONV y ejecutamos:
Entonces introduciremos los datos uno tras otro:

```
RAD XYZ HEX R= 'X' PRG
[HOME LIBRO] 09:36,FEB:10
Diametros en
pulgadas
...10 12 18 20 24 28 32
CONV PRES E1
```

Y la salida será:

```
RAD XYZ HEX R= 'X' PRG
[HOME LIBRO] 09:37,FEB:10
3:
2:
1: ( .1524 .2032 .254
     .3048 .4572 .508
     .6096 .7112 .8128 )
CONV PRES E1
```

COMENTARIOS:

1. Notemos que los datos van uno tras otro y separados por espacios
2. Una vez puestos los datos en la pila se ejecuta el comando **DEPTH** el cual pone en la pila el numero de datos que se encuentran en esta, recordemos que tenemos que especificar el numero de datos que hay en la pila para formar la lista de lo contrario se producirá un error. Luego viene el comando **→LIST** que nos forma una lista con los datos que están en la pila y luego viene la operación para convertir de pulgadas a metros $0.0254 *$, es decir multiplicamos por 0.0254 (Fijémonos que esta en **RPN** lo cual aquí es absolutamente necesario)
3. Observemos que la operación ($0.0254 *$) se efectuó sobre todos los elementos de la lista lo cual es una gran ventaja.
4. Es necesario hacerle **DEBUG** a este código para así poder ver la ejecución paso a paso. (MUY IMPORTANTE)

Supongamos un segundo ejemplo que en realidad es el mismo anterior pero con una pequeña variación: Mostraremos los resultados con su respectiva unidad. Para esto lo único que haremos será multiplicar al final por 1_m. Veamos el código:

```
« "Diametros en
pulgadas" " " INPUT OBJ→
DEPTH →LIST 0.0254 * 1_m *
»
```

Así lo vemos en la pila:

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 09:40, FEB:10
1: «
  "Diametros en=4pul...
  " " INPUT OBJ→
  DEPTH →LIST .0254 *
  1._m * »
CONV PRES E1

```

Grabamos este código bajo el nombre CON2 y ejecutamos: La salida será la siguiente:

COMENTARIOS:

1. Vemos que no se visualizan todos los datos en pantalla. Para solucionar esto oprimimos la tecla que tiene la flecha hacia abajo (Tercera fila quinta tecla)

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 09:42, FEB:10
1: ( .1524_m .2032_m
  .254_m .3048_m
  .4572_m .508_m
  .6096_m .7112_m
  .8128_m )
CON2 CONV PRES E1

```

2. Fijémonos en el código que hay un espacio entre 1_m y el signo * (por)

3. Una vez mas la operación 1_m * se efectuó sobre todos los elementos de la lista.

Una vez mas espero que se hayan entendido muy bien los ejemplos y en especial como se utilizan los comandos **DEPTH** y **→LIST** fundamentales en la creación de listas.

6.3 ADICIÓN DE OBJETOS A UNA LISTA

Se pueden añadir objetos a una lista después de que esta ya esta creada. Se hace de manera muy fácil y es de la siguiente manera:

Supongamos que tenemos la siguiente lista {A N I T S I R} y queremos adicionarle la letra C.

Lo único que tenemos que hacer es lo siguiente:

- Poner la lista en el nivel dos de la pila.
 - En el nivel uno el elemento que vamos a adicionar y presionar el signo +
- Veamos como se vería en la pila:

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 09:45, FEB:10
5:
4:
3:
2: ( A N I T S I R )
1: 'C'
CON2 CONV PRES E1

```

Y presionamos el signo +. Se vera así:

```
RAD XYZ HEX R= 'X'  
(HOME LIBRO) 09 45 FEB:10  
5:  
4:  
3:  
2:  
1: ( A N I T S I R C )  
CON2 CONV PRES E1
```

COMENTARIOS:

1. Aparentemente es una operación sencilla y en efecto lo es. Pero el objetivo es que tengan muy en cuenta como se adiciona un elemento a una lista ya que mas adelante retomaremos bastante este concepto.

6.4 COMANDOS QUE PERMITEN OPERAR SOBRE LISTAS

Como se dijo anteriormente, el buen manejo de listas es quizás uno de los elementos de programación más importante que hay, dado que las listas son muy flexibles para manejar y operar sobre datos dentro de un programa.

Ahora veremos los comandos que permiten operar sobre las listas. Estos comandos nos permiten extraer elementos de una lista, sumar los elementos que componen una lista, etc. Vamos entonces a explicar estos comandos.

6.4.1 ΔLIST

Este comando determina la diferencia entre dos elementos consecutivos de una lista. Por ejemplo si tenemos una lista con los siguientes elementos { 2 4 6 6 } lo que hace es devolvernos la lista { 2 2 0 }

Veamos un segundo ejemplo:

Consideremos la siguiente lista y determinemos las diferencias de los elementos consecutivos:

```
RAD XYZ HEX R= 'X'  
(HOME LIBRO) 09:47 FEB:10  
5:  
4:  
3:  
2:  
1: ( 1 2 3 5 8 9 )  
+RTB +OTB EC ED TED VV
```

Ahora ejecutamos el comando ΔLIST y veremos lo siguiente:

```

RAD XYZ HEX R= 'X'
[HOME LIBR0] 09:48,FEB:10
5:
4:
3:
2:
1: ( 1 1 2 3 1 )
@LIST @ELIST @MLIST SORT REVL1 ADD

```

COMENTARIOS:

1. Importante resaltar que la diferencia se efectúa de derecha a izquierda.
2. El resultado es arrojado en lista.
3. Es un comando muy sencillo pero de gran utilidad dentro de programas donde estemos manejando listas con muchos datos.

6.4.2 ΣLIST

Este comando nos permite hacer la sumatoria de todos los elementos de una lista y nos arroja el resultado como un objeto numero real. Veamos el siguiente ejemplo:
Consideremos la siguiente lista

```

RAD XYZ HEX R= 'X'
[HOME LIBR0] 09:59,FEB:10
5:
4:
3:
2:
1: ( 1 1 2 3 1 )
@LIST @ELIST @MLIST SORT REVL1 ADD

```

Ejecutamos el comando ΣLIST:

```

RAD XYZ HEX R= 'X'
[HOME LIBR0] 10 00,FEB:10
5:
4:
3:
2:
1: 8
@LIST @ELIST @MLIST SORT REVL1 ADD

```

COMENTARIOS:

Ninguno

Veamos ahora un ejemplo con una lista que contiene STRINGS:
 Consideremos la siguiente lista.

```
RAD XYZ HEX R= 'X'
[HOME LIBRO] 10 03,FEB:10
4:
3:
2:
1: ( "L" "A" "T" "E"
    "M" )
OBJ+ +ARRAY +LIST +STR +TAG +UNIT
```

Ejecutamos el comando Σ LIST:

```
RAD XYZ HEX R= 'X'
[HOME LIBRO] 10:04,FEB:10
5:
4:
3:
2:
1: "LATEM"
@LIST @ELIST @MLIST SORT REVL1 ADD
```

Lo que hizo fue “sumarnos” o concatenarnos los STRINGS que estaban dentro de la lista

6.4.3 Π LIST

Este comando efectúa el producto de todos los elementos de una lista. Veamos el siguiente ejemplo:

Consideremos la siguiente lista

```
RAD XYZ HEX R= 'X'
[HOME LIBRO] 10 09,FEB:10
5:
4:
3:
2:
1: ( 2 3 5 6 )
@LIST @ELIST @MLIST SORT REVL1 ADD
```

Ejecutamos el comando Π LIST:

```
RAD XYZ HEX R= 'X'
[HOME LIBRO] 10 10,FEB:10
5:
4:
3:
2:
1: 180
@LIST @ELIST @MLIST SORT REVL1 ADD
```

COMENTARIOS:
Ninguno

6.4.4 SORT

Este comando ordena todos los elementos de una lista en forma ascendente. Veamos el siguiente ejemplo:

Consideremos la siguiente lista

```
RAD XYZ HEX R= 'X'  
(HOME LIBRO) 10 12,FEB:10  
-----  
4:  
3:  
2:  
1: { 9 8 5 6 7 4 3 8 9  
    }  
@LIST|ELIST|TLIST|SORT|REVL|ADD
```

Ejecutamos el comando **SORT**:

```
RAD XYZ HEX R= 'X'  
(HOME LIBRO) 10:20,FEB:10  
-----  
4:  
3:  
2:  
1: { 3 4 5 6 7 8 8 9 9  
    }  
@LIST|ELIST|TLIST|SORT|REVL|ADD
```

COMENTARIOS:

1. Prueben este comando pero con una lista que contenga STRINGS con letras

6.4.5 REVLIST

Este comando invierte el orden de los elementos de una lista. Veamos el siguiente ejemplo:
Consideremos la siguiente lista

```
RAD XYZ HEX R= 'X'  
(HOME LIBRO) 10 22,FEB:10  
-----  
4:  
3:  
2:  
1: { 3 4 5 6 7 8 8 9 9  
    }  
@LIST|ELIST|TLIST|SORT|REVL|ADD
```

Ejecutamos el comando **REVLIST**:

```

RAD XYZ HEX R= 'X'
[HOME LIBR03] 10:24,FEB:10
4:
3:
2:
1: ( 9 9 8 8 7 6 5 4 3
)
@LIST|ELIST|MLIST|SORT|REVLI|ADD

```

COMENTARIOS:
Ninguno

6.4.6 ADD

Este comando suma el contenido de los elementos de dos listas. Veamos el siguiente ejemplo: Consideremos las siguientes listas:

```

RAD XYZ HEX R= 'X'
[HOME LIBR03] 10:28,FEB:10
3:
2: ( 9 9 8 8 7 6 5 4 3
)
1: ( 9 9 8 8 7 6 5 4 3
)
@LIST|ELIST|MLIST|SORT|REVLI|ADD

```

Ejecutamos el comando **ADD**

```

RAD XYZ HEX R= 'X'
[HOME LIBR03] 10:29,FEB:10
4:
3:
2:
1: ( 18 18 16 16 14 12
10 8 6 )
@LIST|ELIST|MLIST|SORT|REVLI|ADD

```

COMENTARIOS:
Ninguno.

Ahora veremos algunos comandos que nos permiten operar sobre listas pero son un poco mas elaborados y “poderosos”

6.4.6 GET

Este comando nos permite extraer un elemento de una lista dado su índice de posición. Para entender mejor veamos el siguiente ejemplo: Consideremos la siguiente lista la cual contiene tres STRINGS

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 10 32,FEB:10
-----
4:
3:
2:
1: { "SIRC" "ANDR"
      "KCUF" }
[LIST|ELIST|MLIST|SORT|REVI|ADD]

```

Si queremos extraer el primer elemento de la lista entonces ponemos en la pila la lista y el índice de posición del elemento a extraer, en este caso 1

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 10 34,FEB:10
-----
4:
3:
2: { "SIRC" "ANDR"
      "KCUF" }
1: 1
[LIST|ELIST|MLIST|SORT|REVI|ADD]

```

Ejecutamos el comando **GET**:

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 10 34,FEB:10
-----
5:
4:
3:
2:
1: "SIRC"
[GET|GETI|PUT|PUTI|SIZE|POS]

```

COMENTARIOS:
Ninguno

6.4.7 PUT

Este comando nos permite poner un elemento dentro de una lista dados su índice de posición y el objeto a introducir. Para entender mejor veamos el siguiente ejemplo:
Consideremos la siguiente lista:

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 10:37,FEB:10
-----
4:
3:
2:
1: { "A" "N" "I" "T"
      "S" "I" "C" }
[GET|GETI|PUT|PUTI|SIZE|POS]

```


Entonces vamos a introducir el STRING "R" en la posición siete. Colocamos en la pila lo siguiente:

- La lista
- El índice de posición (en este caso siete)
- STRING que vamos a poner

```
RAD XYZ HEX R= 'X'  
{HOME LIBRO} 10 42,FEB:10  
-----  
4:  
3: { "A" "N" "I" "T"  
    "S" "I" "C" }  
2:  
1: "R"  
-----  
W | CON2 | CONV | PRES | E1 |
```

Ejecutamos el comando **PUT**:

```
RAD XYZ HEX R= 'X'  
{HOME LIBRO} 10 42,FEB:10  
-----  
4:  
3:  
2:  
1: { "A" "N" "I" "T"  
    "S" "I" "R" }  
-----  
GET | GETI | PUT | PUTI | SIZE | POS |
```

COMENTARIOS:

Fijémonos que el elemento que estaba en la posición siete (C) fue sustituido por R

6.4.8 SIZE

Este comando nos permite saber la cantidad de elementos que componen una lista. Consideremos la siguiente lista:

```
RAD XYZ HEX R= 'X'  
{HOME LIBRO} 10 44,FEB:10  
-----  
4:  
3:  
2:  
1: { "A" "N" "I" "T"  
    "S" "I" "R" }  
-----  
GET | GETI | PUT | PUTI | SIZE | POS |
```

Ejecutamos el comando **SIZE**:

```
RAD XYZ HEX R= 'X'  
{HOME LIBRO} 10 45,FEB:10  
-----  
5:  
4:  
3:  
2:  
1: 7.  
-----  
GET | GETI | PUT | PUTI | SIZE | POS |
```

COMENTARIOS:

Ninguno

6.4.9 POS

Este comando nos permite conocer el índice de posición dentro de una lista, de un elemento dado. Consideremos el siguiente ejemplo:

```
RAD XYZ HEX R= 'X'  
[HOME LIBRO] 10 47 FEB:10  
-----  
4:  
3:  
2:  
1: ( "A" "N" "I" "T"  
    "S" "I" "R" )  
GET GETI PUT PUTI SIZE POS
```

Queremos saber cual es el índice de posición del STRING T. Entonces:

```
RAD XYZ HEX R= 'X'  
[HOME LIBRO] 10 49 FEB:10  
-----  
4:  
3:  
2: ( "A" "N" "I" "T"  
    "S" "I" "R" )  
1: "T"  
GET GETI PUT PUTI SIZE POS
```

Ejecutamos el comando **POS**:

```
RAD XYZ HEX R= 'X'  
[HOME LIBRO] 10:50 FEB:10  
-----  
5:  
4:  
3:  
2:  
1: 4.  
GET GETI PUT PUTI SIZE POS
```

COMENTARIOS:

1. Inicialmente podemos pensar que no tiene ninguna aplicación funcional. Al terminar la explicación de cada uno de los comandos, haremos un ejemplo donde nos daremos cuenta la importancia de este comando.

6.4.10 HEAD

Este comando nos devuelve el primer elemento (cabeza) de una lista. Consideremos el siguiente ejemplo:

```

RAD XYZ HEX R= 'X'
<HOME LIBR0> 10:51,FEB:10
4:
3:
2:
1: { "A" "N" "I" "T"
    "S" "I" "R" }
GET GETI PUT PUTI SIZE POS

```

Si ejecutamos el comando **HEAD** nos devolverá lo siguiente:

```

RAD XYZ HEX R= 'X'
<HOME LIBR0> 10:52,FEB:10
5:
4:
3:
2:
1: "A"
HEAD TAIL LIST

```

COMENTARIOS:

Ninguno

6.4.11 TAIL

Este comando nos elimina el primer elemento de una lista. Considerando la misma lista anterior nos devolverá lo siguiente:

```

RAD XYZ HEX R= 'X'
<HOME LIBR0> 10:54,FEB:10
4:
3:
2:
1: { "A" "N" "I" "T"
    "S" "I" "R" }
HEAD TAIL LIST

```

```

RAD XYZ HEX R= 'X'
<HOME LIBR0> 10:55,FEB:10
4:
3:
2:
1: { "N" "I" "T" "S"
    "I" "R" }
HEAD TAIL LIST

```

VEAMOS UN EJEMPLO QUE APLICA ALGUNOS DE LOS COMANDOS ANTERIORES:

Este ejemplo es la “esencia” para escribir un programa que emule una agenda de telefónica o como se le quiera llamar, la cual podrá contener todos los aspectos de determinado contacto.

Escribamos entonces un programa que con base en el nombre del contacto nos arroje el numero celular y el correo electrónico.

Grabemos entonces tres listas las cuales contendrán:

1. Nombre.
2. Numero celular
3. e-mail.

NOMBRES:

```
{ "Andrés García" "Carlos Díaz" "Fabián Herrera" "Santiago José López" }
```

Una vez puesta en pila esta lista la grabamos con el nombre **NOMBRES**

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 11 08,FEB:10
2: ( "Andres Garcia"
     "Carlos Diaz"
     "Favian Herrera"
     "Santiago Jose Lop..."
1: 'NOMBRES'
ELEM PROC OBJ+ -LIST SUB REPL

```

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 11 09,FEB:10
5:
4:
3:
2:
1:
NOMBR

```

NÚMEROS CELULARES:

{ "3114853881" "3005711826" "3103178192" "3002105072" }

Una vez puesta en pila esta lista la grabamos con el nombre TELS

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 11 13,FEB:10
2: ( "3114853881"
     "3005711826"
     "3103178192"
     "3002105072" }
1: 'TELS'
C+R R+C DUM CHR DTAG EQ+

```

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 11:13,FEB:10
5:
4:
3:
2:
1:
TELS NOMBR

```

E - MAIL:

{ argarcia35@ucatolica.edu.co "lithus@metallica.com" "fan@megadeth.com" "santi@gcn.com" }

Una vez puesta en pila esta lista la grabamos con el nombre EMAIL

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 11:18,FEB:10
2: (
     "argarcia35@ucatol..."
     "lithus@metallica..."
     "fan@megadeth.com"
1: 'EMAIL'
EDIT VIEW STACK RCL PURGE CLEAR

```

```

RAD XYZ HEX R= 'X'
[HOME LIBRO] 11 18,FEB:10
5:
4:
3:
2:
1:
EMAIL TELS NOMBR

```

Ya teniendo las tres listas grabadas entonces procedemos a escribir el código:

```

«
"Escriba el nombre del
contacto" "" INPUT
CLLCD
'c' STO
c
DUP
1 DISP
NOMBRES
SWAP
POS
DUP

```

Abre Programa

```

Blanqueamos la pantalla
Grabamos el nombre introducido
  Llamamos el nombre a la pila
  Lo duplicamos
Lo colocamos en pantalla
  Llamamos la lista nombres a la pila
Rotamos la pila
  Encontramos la POS de Carlos Díaz
  Duplicamos el STRING "Carlos Díaz"

```

```

TELS
SWAP
GET
2 DISP
EMAIL
SWAP
GET
3 DISP
"HAY "
NOMBRES
SIZE
→STR
+
"CONTACTOS"
+

```

```

Llamamos la lista TELS a la pila
Rotamos la pila
    Extraemos el TEL de la POS 2
Lo colocamos en pantalla
Llamamos la lista EMAIL a la pila
Rotamos la pila
    Extraemos el EMAIL de la POS 2
Lo colocamos en pantalla
Ponemos el STRING HAY en la pila
    Llamamos la lista Nombres a la pila
    Averiguamos su tamaño
Lo convertimos en STRING
    Concatenamos los 2 STRINGS
    Ponemos el STRING CONTACTOS en la pila
    Lo concatenamos al STRING que hay en pila

```

```

5 DISP
0 WAIT
DROP
{ c } PURGE
»

```

```

Lo colocamos en pantalla
Esperamos a que se pulse una tecla
Borramos la "semilla" dejada por la tecla pulsada
Borramos de la memoria la VAR c
Cierra Programa

```

Grabamos este código con el nombre **INICI** y lo ejecutamos para Carlos Díaz por ejemplo:

```

RAD XYZ HEX R= 'X'          PRG
[HOME LIBRO]
-----
Escriba el nombre del
contacto

Carlos Diaz
INICI EMAIL TELS NOMBRE

```

```

Carlos Diaz
3005711826
lithus@metallica.com

HAY 4.CONTACTOS

INICI EMAIL TELS NOMBRE

```

Para Santiago José López:

```

RAD XYZ HEX R= 'X'          PRG
[HOME LIBRO]
-----
Escriba el nombre del
contacto

Santiago Jose Lopez
INICI EMAIL TELS NOMBRE

```

```

Santiago Jose Lopez
3002105072
santi@gcn.com

HAY 4.CONTACTOS

INICI EMAIL TELS NOMBRE

```

COMENTARIOS:

1. Notemos que al llamar las listas que contienen los diferentes datos, lo hacemos sin ninguna clase de delimitador.
2. Al introducir las listas debemos hacerlo en el mismo orden en que aparecen es decir; el elemento 1 de la lista NOMBRES debe corresponder al elemento 1 de la lista TELS y Elemento 1 de lista EMAIL.

3. Este código se puede optimizar muchísimo y lo retomaremos mas adelante, pero para efectos del ejemplo esta bien así. Por ultimo recomiendo **muchísimo** hacerle un **DEBUG** a este código si algo no quedo muy claro.

Vamos a un segundo ejemplo:

Para calcular la capacidad de carga de un suelo aplicando la teoría y las ecuaciones propuestas por su eminencia Dr. KARL VON TERZAGHI se encuentran dentro de dichas ecuaciones, unos coeficientes llamados FACTORES DE CAPACIDAD DE CARGA (N_c , N_q y N_γ) los cuales son función del ángulo de fricción interna (ϕ .) Estos coeficientes se pueden calcular mediante una ecuación pero se hace extenso ya que hay que conocer muchos otros datos.

A continuación escribiremos un programa que nos pregunta únicamente el ángulo de fricción interna y nos devuelve los factores de capacidad de carga (N_c , N_q y N_γ)

Estos factores se encuentran en CIMENTACIONES de Braja M Das. **Ojo**, en el de Cimentaciones no el de fundamentos de Ingeniería Geotécnica.

Primero que todo debemos grabar nuestros coeficientes dentro de una lista. Una lista para los valores de ϕ , una lista para los valores de N_c , una lista para los valores de N_q y otra lista para los valores de N_γ .

Las listas son las siguientes:

VALORES DE ϕ :

```
{ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38
39 40 41 42 43 44 45 46 47 48 49 50 }
```

Una vez puesta en pila esta lista la grabamos con el nombre ϕ

```
RAD XYZ HEX R= 'X'
<HOME LIBRO>
2: { 0 1 2 3 4 5 6 7 8
    9 10 11 12 13 14 15
    16 17 18 19 20 21
    22 23 24 25 26 27
1: '0'
OBJ+ +ARRAY +LIST +STR +TAG +UNIT
```

```
RAD XYZ HEX R= 'X'
<HOME CP>
5:
4:
3:
2:
1:
p
```

VALORES DE N_c :

```
{ 5.7 6 6.3 6.62 6.97 7.34 7.73
8.15 8.6 9.09 9.61 10.16 10.76
11.41 12.11 12.86 13.68 14.6 15.12
16.56 17.69 18.92 20.27 21.75
23.36 25.13 27.09 29.24 31.61
34.24 37.16 40.41 44.04 48.09
52.64 57.75 63.53 70.01 77.5
85.97 95.66 106.81 119.67
134.58 151.95 172.28 196.22
224.55 258.28 298.71 347.5 }
```

Una vez puesta en pila esta lista la grabamos con el nombre N_c

```

RAD XYZ HEX R= 'X'
[HOME CP]
2: { 5.7 6 6.3 6.62
      6.97 7.34 7.73 8.15
      8.6 9.09 9.61 10.16
      10.76 11.41 12.11
1: 'Nc'
OBJ+ +ARRAY +LIST +STR +TAG +UNIT

```

```

RAD XYZ HEX R= 'X'
[HOME CP]
5:
4:
3:
2:
1:
Nc p

```

VALORES DE Nq:

```

{ 1 1.1 1.22 1.35 1.49 1.64
  1.81 2 2.21 2.44 2.69 2.98
  3.29 3.63 4.02 4.45 4.92 5.45
  6.04 6.7 7.44 8.26 9.19 10.23
  11.4 12.72 14.21 15.9 17.81
  19.98 22.46 25.28 28.52 32.23
  36.5 41.44 47.16 53.8 61.55
  70.61 81.27 93.85 108.75 126.5
  147.74 173.28 204.19 241.8
  287.85 344.63 415.14 }

```

Una vez puesta en pila esta lista la grabamos con el nombre Nq

```

RAD XYZ HEX R= 'X'
[HOME CP]
2: { 1 1.1 1.22 1.35
      1.49 1.64 1.81 2
      2.21 2.44 2.69 2.98
      3.29 3.63 4.02 4.45
1: 'Nq'
Nc p

```

```

RAD XYZ HEX R= 'X'
[HOME CP]
5:
4:
3:
2:
1:
Nq Nc p

```

VALORES DE Ny:

```

{ 0 .01 .04 .1 .14 .2 .27
  .35 .44 .56 .69 .85 1.04 1.26
  1.52 1.82 2.18 2.59 3.07 3.64
  4.31 5.09 6 7.08 8.34 9.84
  11.6 13.7 16.18 19.13 22.65
  26.87 31.94 38.04 45.41 54.36
  65.27 78.61 95.03 115.31 140.51
  171.99 211.56 261.6 325.34
  407.11 512.84 650.67 831.99
  1072.8 }

```

Una vez puesta en pila esta lista la grabamos con el nombre Ny

```

RAD XYZ HEX R= 'X'
[HOME CP]
2: { 0 .01 .04 .06 .1
      .14 .2 .27 .35 .44
      .56 .69 .85 1.04
      1.26 1.52 1.82 2.18
1: 'Ny'
OBJ+ +ARRAY +LIST +STR +TAG +UNIT

```

```

RAD XYZ HEX R= 'X'
[HOME CP]
5:
4:
3:
2:
1:
Ny Nq Nc p

```

Ya teniendo las tres listas grabadas entonces procedemos a escribir nuestro programa:

```

« "Valor de Ø" ""
INPUT OBJ→
'FI' STO      Grabamos el valor de Ø con el nombre FI
Ø             Llamamos la lista que contiene los valores de Ø
FI POS       Determinamos la posición del valor de Ø
FI2 STO     Grabamos la posición del valor de Ø
Ø             Llamamos la lista que contiene los valores de Ø
FI2 GET     Sacamos el valor de Ø que esta en la posición FI2
'Ø' →TAG    Lo etiquetamos
Nc           Llamamos la lista que contiene los valores de Nc
FI2 GET     Sacamos el valor de Nc que esta en la posición FI2
'Nc' →TAG   Lo etiquetamos
Nq           Llamamos la lista que contiene los valores de Nq
FI2 GET     Sacamos el valor de Nq que esta en la posición FI2
'Nq' →TAG   Lo etiquetamos
Nγ          Llamamos la lista que contiene los valores de N'
FI2 GET     Sacamos el valor de Nγ que esta en la posición FI2
'N' →TAG    Lo etiquetamos
{ FI2 FI } PURGE Eliminamos las variables FI2 y FI
»

```

Grabamos el código con el nombre **AQUÍ** y ejecutamos para un ángulo de 23°:

```

RAD XYZ HEX R= 'X'      PRG
[HOME CP]
Valor de Ø

23
AQUI  Nγ  Nq  Nc  Ø

```

```

RAD XYZ HEX R= 'X'
[HOME CP]
Ø:
4:
00:      Nc: 21.75
2:      Nq: 10.23
1:      Nγ: 6
AQUI  Nγ  Nq  Nc  Ø

```

COMENTARIOS:

1. Notemos que al llamar las listas que contienen los valores, lo hacemos sin ninguna clase de delimitador.
2. Para poner la etiqueta si debemos hacerlo con los delimitadores " de lo contrario nos vertería todo el contenido de la lista a la pila.
3. Este código se puede optimizar muchísimo. Pero para efectos del ejemplo esta bien así. Por ultimo recomiendo hacerle un **DEBUG** si algo no quedo muy claro.

Bueno, ahora sigamos explicando los demás comandos que permiten operar sobre listas.

6.4.12 DOLIST

Este comando nos permite operar sobre un grupo de listas. Veamos un ejemplo: Tenemos las siguientes listas en la pila:


```

RAD XYZ HEX R= 'X'
[HOME]
5:
4:
3:
2:      ( 1 23 4 5 )
1:      ( 2 1 3 5 )
CP LIBRO L ESTRU VX

```

Vamos a aplicar la operación “potenciación”. Esta operación debe ir dentro de delimitadores de programa así:

```

RAD XYZ HEX R= 'X'
[HOME]
5:
4:
3:      ( 1 23 4 5 )
2:      ( 2 1 3 5 )
1:      ( 2 1 3 5 )
CP LIBRO L ESTRU VX

```

Ejecutamos el comando DOLIST:

```

RAD XYZ HEX R= 'X'
[HOME]
5:
4:
3:
2:
1:      ( 1 23 64 3125 )
DOLIS DOSUB NSUB ENDSU STREA REVL

```

COMENTARIOS:
NINGUNO

6.4.13 DOSUBS

Este comando es supremamente funcional y lo que hace es ejecutar varias operaciones (no-solo una) a todos los elementos de una lista. Veamos un ejemplo:

Consideremos la siguiente lista:

```

RAD XYZ HEX R= 'X'
<HOME>
5:
4:
3:
2:
1:      ( 1 2 3 4 5 )
DOLIS DOSUB NSUB ENDSU STREA REVL

```

Especificamos un tamaño de paso igual a 1. Es decir que aplique las operaciones a cada elemento de la lista

```

RAD XYZ HEX R= 'X'
<HOME>
5:
4:
3:
2:      ( 1 2 3 4 5 )
1:      1
DOLIS DOSUB NSUB ENDSU STREA REVL

```

Vamos a aplicar las siguientes operaciones las cuales deben ir dentro de delimitadores de programa.

```

RAD XYZ HEX R= 'X'
<HOME>
5:
4:
3:      ( 1 2 3 4 5 )
2:      1
1:      « 2 ^ 3 + √ »
DOLIS DOSUB NSUB ENDSU STREA REVL

```

Ejecutamos el comando **DOSUBS**:

```

RAD XYZ HEX R= 'X'
<HOME>
2:
1: ( 2. 2.64575131106
    3.46410161514
    4.35889894354
    5.29150262213 )
DOLIS DOSUB NSUB ENDSU STREA REVL

```

Veamos un segundo ejemplo:

Consideremos la siguiente lista:

```
RAD XYZ HEX R= 'X'
<HOME>
5:
4:
3:
2:
1:          ( 1 2 3 4 5 )
DOLIS DOSUB NSUB ENDSU STREA REVL
```

Ahora especifiquemos un tamaño de paso igual a 2

```
RAD XYZ HEX R= 'X'
<HOME>
5:
4:
3:
2:          ( 1 2 3 4 5 )
1:                                     2
DOLIS DOSUB NSUB ENDSU STREA REVL
```

Con la siguiente operación

```
RAD XYZ HEX R= 'X'
<HOME>
5:
4:
3:          ( 1 2 3 4 5 )
2:                                     2
1:          << + >>
DOLIS DOSUB NSUB ENDSU STREA REVL
```

Ejecutamos **DOSUBS**:

```
RAD XYZ HEX R= 'X'
<HOME>
5:
4:
3:
2:
1:          ( 3 5 7 9 )
DOLIS DOSUB NSUB ENDSU STREA REVL
```

Lo que hizo fue sumar los elemento adyacentes entre si.

COMENTARIOS:

Ninguno

Creo que de a esta instancia del libro ustedes ya están en capacidad de seguir examinando que hace y como operan cada uno de los comandos guiándose por el manual de manejo, por lo que de ahora en adelante no haré referencia a los comandos como tal sino solamente a lo que es la programación.

```
RAD XYZ HEX R= 'X'  
CHOME3  
-----  
ESTRUCTURA DE  
VARIABLE LOCAL  
-----  
A | N | G | A | R | 12
```

7. ESTRUCTURA DE VARIABLE LOCAL.

7. ESTRUCTURA DE VARIABLE LOCAL.

7.1 DEFINICIÓN

Las variables locales son variables provisionales creadas por un programa. Existen mientras se ejecuta el programa y a diferencia de las variables globales no se pueden utilizar por fuera del programa a menos que la subrutina este anidada o embebida dentro del proceso de definición de variable local, esto lo entenderán mejor mas adelante.

Para entender mejor como se crean y utilizan las variables locales veamos lo siguiente.

7.2 SINTAXIS DE VARIABLE LOCAL DENTRO DE UN PROGRAMA

Para entender veamos el siguiente ejemplo:

Escribiremos un programa que calcula el coeficiente de empuje activo en muros de retención en voladizo. La ecuación (entre otras) que se utiliza es la siguiente:

$$K_a = \cos(\alpha) * (\cos(\alpha) - \sqrt{ \cos(\alpha)^2 * \cos(\phi)^2 }) / (\cos(\alpha) + \sqrt{ \cos(\alpha)^2 * \cos(\phi)^2 })$$

Donde:

α = Angulo entre la corona del muro y el talud natural.

ϕ = Angulo de fricción interna del suelo de relleno

Los datos que necesitamos son entonces α y ϕ . Veamos el código y luego lo explico.

```
« "Valor de  $\alpha$ " ""
```

```
INPUT OBJ→
```

```
"Valor de  $\phi$ " ""
```

```
INPUT OBJ→
```

```
→  $\alpha$   $\phi$ 
```

```
« 'COS( $\alpha$ )*(COS( $\alpha$ )-f(COS( $\alpha$ )^2-COS( $\phi$ )^2))/
```

```
(COS( $\alpha$ )+f(COS( $\alpha$ )^2-COS( $\phi$ )^2))' EVAL
```

```
» 2 RND
```

```
→STR
```

```
"Ka= "
```

```
SWAP +
```

```
CLLCD MSGBOX
```

```
»
```

En este punto del libro ya conocemos gran parte de este código. Aquí lo único nuevo es la definición de variable local.

En este ejemplo teníamos que definir dos variables locales (α y ϕ). Para definir estas dos utilizamos el símbolo →.

Después del símbolo vienen los nombres con los cuales vamos a nombrar las variables. En este caso eran α y ϕ .

Nótese que después de nombrar las dos variables tenemos que abrir otros delimitadores de programa.

Después de esto viene la operación y luego el comando RND el cual nos trunca la respuesta a dos posiciones decimales.

El resto del código ya lo conocemos perfectamente. Importante aclarar que las variables locales no se pueden volver a llamar después de haber cerrado los delimitadores de programa que abrimos después de definir las dos variables locales (→ α ϕ)

Recomiendo hacer un DEBUG a este código si no se entendió algo.

7.3 ESTRUCTURA DE VARIABLE LOCAL DENTRO DE UNA SUBRUTINA

Las variables locales también las podemos utilizar dentro de una subrutina siempre y cuando la subrutina este anidada dentro del procedimiento de variable local. Esto quiere decir que la subrutina debe estar dentro de los “segundos” delimitadores de programa que abrimos haciendo referencia al ejemplo anterior.

Veamos el mismo ejemplo anterior pero ahora supondremos que necesitamos una subrutina. La subrutina la llamaremos SUBR.

Supondremos también que la subrutina la utilizaremos únicamente para elevar al cuadrado las dos variables.

```
« "Valor de  $\alpha$ " ""
INPUT OBJ→
"Valor de  $\emptyset$ " ""
INPUT OBJ→
→ ← $\alpha$  ← $\emptyset$ 
  « SUBR
'COS(← $\alpha$ )*(COS(← $\alpha$ )-f(COS(← $\alpha$ )2-COS(← $\emptyset$ )2))/
(COS(← $\alpha$ )+f(COS(← $\alpha$ )2-COS(← $\emptyset$ )2))' EVAL
  » 2 RND
  →STR
  "Ka= "
  SWAP +
  CLLCD MSGBOX
  »
```

El código de la subrutina es el siguiente:

```
« '← $\alpha$ 2' EVAL
'← $\emptyset$ 2' EVAL
»
```

COMENTARIOS:

1. Para definir las variables locales debemos anteponerles el símbolo ← Esto es para poder llamar las variables dentro de la subrutina **SUBR**.
2. La subrutina **SUBR** puede estar antes o después de la operación. Esto depende de las solicitaciones del programa, en este caso puede estar antes o después.
3. En la operación las variables deben aparecer con el símbolo ← de lo contrario no las reconoce.
4. En resumen lo que hace este programa después de haber introducido los datos es ir a la subrutina, ejecutar las operaciones que hay en la subrutina **SUBR** luego sigue con la ejecución del programa, es decir con lo que hay después de **SUBR**

El siguiente es un ejemplo de aplicación de variable locales dentro de una subrutina el cual sirve para calcular empates por línea de energía en régimen supercrítico en aguas negras. Dependiendo del valor de la operación ' $0.319 * Q / \emptyset^{2.5}$ ' se va para una subrutina llamada E2 si el valor es mayor que 0.62; o para una subrutina llamada E3 si el valor de la operación es menor que 0.62.

En este código hay algunos comandos que no han sido explicados en este libro pero que más adelante entenderán. Lo incluyo para que se vea la importancia de las variables locales y como llamarlas dentro de una subrutina.

La variable con la que se inicia el programa es E1:

```

"Caudal de diseño del
tubo de salida
(En m^3/s)" "" INPUT OBJ→
"Diametro del tubo de
salida (En in)" "" INPUT OBJ→
.0254 * ←K ←Q ←ø
« '.319*←Q/←ø^2.5' EVAL 2 RND
'oP' STO
CASE 'oP>.62'
THEN CLLCD "0.319*Q/ø^2.5=
" oP →STR
{ oP } PURGE + 1 DISP
"El empate es sumergido
Presione enter para
solucionar" 3 DISP 0 WAIT DROP E2
END 'oP≤.62'
THEN CLLCD "0.319*Q/ø^2.5= "
oP →STR
{ oP } PURGE + 1 DISP
"El empate es no
sumergido. Presione
ENTER para solucionar" 3 DISP 0 WAIT CLEAR
E3
END
END
» E4 -40 SF
»

```

```
RAD XYZ HEX R= 'X'  
<HOME>  
  
      FLAGS  
  
A   D   G   A   R   12
```

8. FLAGS

8. FLAGS.

8.1 DEFINICIÓN.

Un FLAG o bandera es un indicador de sistema los cuales se pueden anular, fijar o probar. Por ejemplo el FLAG -40 es el indicador de sistema del reloj que nos permite activarlo o desactivarlo, el FLAG -2 es el indicador de sistema de las constantes el cual nos permite poner constantes de manera numérica o simbólica.

8.2 COMO MODIFICAR FLAGS DENTRO DE UN PROGRAMA

Para activar o desactivar FLAGS utilizamos las palabras SF Y CF. SF para activar y CF para desactivar. Veamos un ejemplo donde tenemos que desactivar el reloj para que no nos dañe la presentación de texto en pantalla:

Primero veamos el código y la presentación sin desactivar el reloj:

```
« CLLCD  
" ELIMINADOR DE BASURA
```

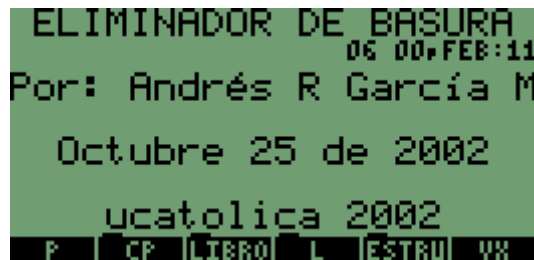
Por: Andrés R García M

Octubre 25 de 2002

```
ucatolica 2002"  
0 DISP 0 WAIT CLEAR
```

»

El cual al ejecutarlo nos muestra lo siguiente:



```
ELIMINADOR DE BASURA  
06 00 FEB:11  
Por: Andrés R García M  
Octubre 25 de 2002  
ucatolica 2002  
P | CF | LIBRO | L | ESTRU | VX
```

Como nos podemos dar cuenta el reloj nos esta “dañando” nuestra presentación ya que no desactivamos el reloj, mediante el FLAG -40.

Ahora veamos el código desactivando el reloj:

```
« CLLCD -40 CF  
" ELIMINADOR DE BASURA
```

Por: Andrés R García M

Octubre 25 de 2002

```
ucatolica 2002"  
0 DISP 0 WAIT CLEAR
```

»

El cual al ejecutarlo nos muestra lo siguiente:

```
ELIMINADOR DE BASURA
Por: Andrés R García M
Octubre 25 de 2002
ucatolica 2002
P CP LIBRO L ESTAU VX
```

COMENTARIOS:

1. Al desactivar el reloj mejoramos mucho la presentación.
2. De esta misma manera se activan o desactivan los demás FLAGS. Nótese que el numero de FLAG es negativo.
3. Los FLAGS son muy importantes y hay que tener especial atención en programas que trabajan con constantes numéricas o simbólicas ya que hay que activar el FLAG antes de cualquier cosa. Lo mismo en programas que trabajan formulas u operaciones matemáticas de manera simbólica. Para una lista completa de los FLAGS refiérase al manual de usuario Pagina O APÉNDICE D1.
4. En el ejemplo anterior para volver a activar el reloj entonces al final ponemos -40 SF. Veamos el código:

```
« CLLCD -40 CF
" ELIMINADOR DE BASURA
```

Por: Andres R Garcia M

Octubre 25 de 2002

```
ucatolica 2002"
0 DISP 0 WAIT CLEAR
-40 SF
```

»



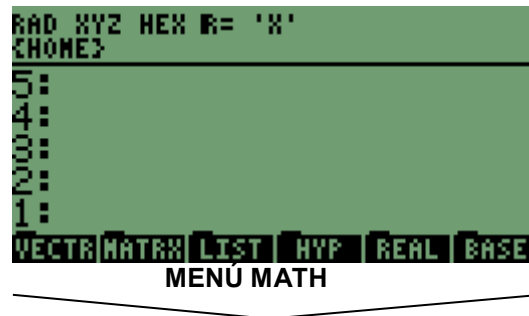
9. MENÚS.

9. MENÚS.

9.1 NÚMEROS DE ASIGNACIÓN DE LOS MENÚS

Todos los menús que ofrece la HP tienen un número de asignación. Como por ejemplo si pulsamos el número 3 y luego MENÚ nos enviara al menú MTH.

Como podemos ver a continuación:



Veamos un ejemplo en donde vamos a entrar dos datos pero queremos que durante esta entrada no aparezca nada en las teclas de menú y al finalizar la entrada nos aparezca el menú actual, es decir el del directorio donde estamos:

Vemos el código:

```
« 1 MENU
"Valor de a?" ""
INPUT OBJ→
"Valor de b?" ""
INPUT OBJ→
2.01 MENU
»
```

El cual al ejecutarlo nos muestra:



```

RAD XYZ HEX R= 'X'
[HOME]
5:
4:
3:
2:
1:
R CP LIBRO L ESTRU V8

```

COMENTARIOS:

1. Como se puede ver durante la entrada de los datos no muestra nada en las teclas de menú. Esto porque antes de iniciar la entrada de los datos pusimos **1 MENU**.
2. Al final de la entrada de los datos nos muestra el menú actual. Esto porque pusimos **2.01 MENU** el cual nos permite ver el menú actual.

9.2 COMO CREAR MENÚS TEMPORALES

Los menús temporales son menús creados directamente por el usuario. Estos menús se pueden crear de dos formas: Mediante la tecla **MENÚ** o mediante el comando **TMENU**.

9.2.1 MEDIANTE LA TECLA MENÚ

Para crear menús personalizados mediante CST debemos hacer los siguiente:

1. Introducir una lista en la pila que a su vez contiene otras " sublistas " las cuales contienen dos argumentos: La etiqueta (es decir lo queremos que aparezca en el menú) y la acción que queremos que se ejecute cuando pulsemos la tecla de menú.

Veamos un ejemplo:

Crearemos un menú temporal mediante la tecla **MENÚ** con la etiqueta SENO la cual ejecuta la función SEN:

```

RAD XYZ HEX R= 'X'
[HOME]
5:
4:
3:
2:
1: ( ( "SENO" SIN ) )
MENU CST TMENU RCLME MODES

```



Entonces veremos lo siguiente:

```

RAD XYZ HEX R= 'X'
{HOME}
-----
5:
4:
3:
2:
1:
SENO

```

Si colocamos un numero en la pila y ejecutamos entonces veremos:

```

RAD XYZ HEX R= 'X'
{HOME}
-----
5:
4:
3:
2:
1: 45
SENO

```

```

RAD XYZ HEX R= 'X'
{HOME}
-----
5:
4:
3:
2:
1: .850903524534
SENO

```

COMENTARIOS:

1. Si volvemos al menú donde estábamos antes de ejecutar la tecla MENU vemos que se ha creado la variable **CST**.

```

RAD XYZ HEX R= 'X'
{HOME}
-----
5:
4:
3:
2:
1: .850903524534
CST R CP LIBRO L ESTRU

```

9.2.2 MEDIANTE EL COMANDO TMENU

Crear menús mediante este comando se hace de manera similar al comando anterior. Para entender como se crean menús temporales mediante **TMENU** veamos el siguiente ejemplo:

Asignaremos 5 etiquetas de menú que contienen los números del 1 al 3 cuyas acciones son poner su correspondiente numero en pantalla mediante el comando **DISP** y una quinta cuya acción es volver al directorio donde nos encontramos:

```

« { { 1
« CLLCD UNO →STR
1 DISP 3 FREEZE
»
}
{2
« CLLCD DOS →STR
2 DISP 3 FREEZE
»
}

```

```

}
{3
« CLLCD TRES →STR
3 DISP 3 FREEZE
»
}
{ } { }
{ ATRAS
« 2.01 MENÚ
CLLCD "FIN" 4 DISP 0 WAIT DROP
2.01 MENU
»
} }
TMENU
»
Lo cual nos muestra:

```



Si pulsamos la tecla de menú 1 hará lo siguiente:



Si pulsamos la tecla de menú 2 hará lo siguiente:



Si pulsamos la tecla de menú 3 hará lo siguiente:



Si pulsamos la tecla de menú **ATRÁS** hará lo siguiente:



COMENTARIOS:

1. Observemos que la acción a ejecutar esta dentro de delimitadores de programa.
2. Para dejar una tecla de menú en blanco la lista debe ir en blanco { }


```
RAD XYZ HEX R= 'X'  
{HOME}  
  
ESTRUCTURAS DE  
PROGRAMACION  
  
A | N | G | A | R | 12
```

10. ESTRUCTURAS DE PROGRAMACION

10. ESTRUCTURAS DE PROGRAMACIÓN.

10.1 DEFINICIÓN

Una estructura de programación permite que un programa decida la tarea a ejecutar dependiendo de las condiciones existentes o de los valores de argumento concreto. Una buena utilización de estas estructuras permite crear programas extraordinariamente flexibles.

10.2 ESTRUCTURAS CONDICIONALES

Las estructuras condicionales permiten que un programa tome una decisión basada en el resultado de una o más pruebas.

10.2.2 IF ...THEN...ELSE...END

Para entender exactamente como funciona esta estructura veamos el siguiente ejemplo:

En alcantarillados de aguas negras, el empate del tubo que entra y del tubo que sale del pozo puede ser sumergido o no sumergido. Sabemos que es sumergido o no sumergido según el resultado de la siguiente operación:

$$0.319 * Q / \Phi^{2.5}$$

Donde :

Q = Caudal de diseño del tubo de salida.

Φ = Diámetro del tubo de salida.

Si el resultado de esta operación es mayor que 0.62 el empate es sumergido.

Si el resultado es menor o igual a 0.62 el empate es no sumergido.

Escribamos entonces un programa que de acuerdo al resultado de la operación $0.319 * Q / \Phi^{2.5}$ nos diga si el empate es sumergido o no sumergido.

Veamos entonces el código:

« 1 MENÚ	Ponemos etiquetas de menú en blanco
"Caudal de diseño del tubo de salida (En m ³ /s)"	Entrar el caudal de diseño
"" INPUT OBJ→	
"Diámetro del tubo de salida (En in)" ""	Entrar el diámetro en pulgadas
INPUT OBJ→	
0.0254 *	Por 0.0254 para pasar a m
→ Q D	Definimos variables locales
« '0.319*Q/D^2.5'	
EVAL 2 RND	Evaluamos la operación y cortamos a dos lugares decimales
»	
→ OP	Definimos una nueva variable local para el resultado de la operación
con el nombre OP	
«	
IF 'OP≤0.62'	Si OP es menor o igual a 0.62
THEN -40 CF	entonces quitamos el reloj de la pantalla
CLLCD	Ponemos en blanco la pantalla
"El empate es no sumergido"	
1 DISP	Ubicamos el texto anterior en el nivel 1
0 WAIT	Se ve el texto hasta que se pulsa una tecla

CLEAR	Borramos la semilla dejada cuando se pulsa la tecla	
ELSE	De lo contrario (Es decir sino es menor o igual a	0.62)
-40 CF	Quitamos el reloj de la pantalla	
CLLCD	Ponemos en blanco la pantalla	
"El empate es sumergido"		
1 DISP	Ubicamos el texto anterior en el nivel 1	
0 WAIT	Se ve el texto hasta que se pulsa una tecla	
CLEAR	Borramos la semilla dejada cuando se pulsa la tecla	
END	FIN	
» -40 SF	Volvemos a poner el reloj	
2.01 MENÚ	Ponemos las etiquetas de MENÚ del directorio actual	
»		

Ejecutemos este código entonces con los siguientes datos:

$Q = 0.56 \text{ m}^3 / \text{s}$

$\Phi = 24''$

```

RAD XYZ HEX R= 'X'          PRG
CHOME3
-----
Caudal de diseño=
del tubo de salida=
(En m^3/s)

0.56
0 | CP | LIBRO | L | ESTRU | VX

```

```

RAD XYZ HEX R= 'X'          PRG
CHOME3
-----
Diametro del tubo de=
salida (En in)

24
0 | CP | LIBRO | L | ESTRU | VX

```

Pulsamos ENTER:

```

El empate es=
no sumergido

0 | CP | LIBRO | L | ESTRU | VX

```

COMENTARIOS:

1. Esta estructura IF...THEN...ELSE...END permite elegir entre dos opciones nada mas.

10.2.3 CASE...THEN...END

Esta estructura a diferencia del IF...THEN...ELSE...END permite elegir entre varias opciones, me explico; en el ejemplo anterior solo se permitía decidir la acción a ejecutar si el resultado era menor o igual a 0.62 o si era mayor a 0.62.

La estructura **CASE...THEN...END** nos permite entonces elegir entre n opciones.

Para poder explicar un ejemplo competo de **CASE...THEN...END** veamos primero un tema de vital importancia en el desarrollo de programas completos: La organización del teclado.

10.2.3.1 ORGANIZACIÓN DEL TECLADO

Cada tecla de la HP tiene un numero de asignación. Así por ejemplo si estamos esperando que se pulse una tecla y pulsamos la tecla **ENTER** aparecerá el numero **105.1**. Veamos un ejemplo:

Escribamos el siguiente código el cual espera que se pulse una tecla.

```
« CLLCD -40 CF
"ESPERO UNA TECLA"
1 DISP 0 WAIT
-40 SF
»
```

Realmente el comando que espera la tecla es el **WAIT**. Este código hará lo siguiente:



Si pulsamos por ejemplo la tecla ENTER aparecerá lo siguiente:



Ya se imaginaron ustedes que dependiendo del número que aparece en la pila es función de la tecla que se pulsa y nos permitirá hacer lo que queramos dependiendo del mismo.

En forma general, el teclado y sus números de asignación se pueden representar de la siguiente manera:

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

X puede tomar cualquier valor entre 0 y 6 dependiendo de si se pulsó una de las siguientes teclas antes:

Tecla de cambio derecha (la flecha verde) Valor de $x = 3$

Tecla de cambio izquierda (la flecha morada) Valor de $x = 2$

Tecla alfa (α) Valor de $x = 4$

Tecla alfa (α) + tecla de cambio derecha: Valor de $x = 6$

Tecla alfa (α) + tecla de cambio izquierda: Valor de $x = 5$

Sin ninguna tecla antepuesta: Valor de $x = 1$

Veamos entonces un ejemplo en donde se **CASE**:

Escribamos un programa o mas bien la presentación de un programa que nos presenta un MENU con cuatro opciones y dependiendo la tecla que pulsemos iniciara cualquiera de las cuatro opciones; y sino es ninguna de las cuatro nos mostrara de nuevo la presentación. Estas opciones serán:

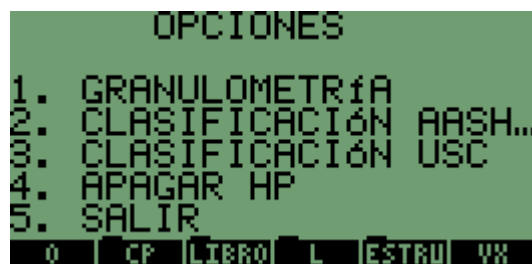
1. GRANULOMETRÍA.
2. CLASIFICACIÓN AASHTO
3. CLASIFICACIÓN USC.
4. APAGAR HP
5. SALIR.

Veamos la primera parte del programa para así entenderlo mejor:

```
« CLLCD                "Blanquea" la pantalla
-40 CF                 Quita el reloj
"  OPCIONES

1. GRANULOMETRÍA
2. CLASIFICACIÓN AASHTO
3. CLASIFICACIÓN USC
4. SALIR"
1 DISP                 Pone le texto en pantalla
0 WAIT                Espera una tecla
»
```

Al ejecutarlo muestra lo siguiente



Pulemos la tecla 1 por ejemplo: Granulometría:

```
RAD XYZ HEX R= 'X'  
{HOME}  
5:  
4:  
3:  
2:  
1: 92.1  
0  CP LIBRO L  ESTRU VX
```

Ahora pulsemos la tecla 2: Clasificación AASHTO

```
RAD XYZ HEX R= 'X'  
{HOME}  
5:  
4:  
3:  
2:  
1: 93.1  
0  CP LIBRO L  ESTRU VX
```

Escribamos entonces todo el código el cual nos permitirá escoger la ruta a seguir en el programa dependiendo de la tecla que se pulso, y si no se pulso una de las cinco teclas esperadas vuelve a ejecutar la presentación:

```
« CLLCD  
-40 CF  
" OPCIONES  
  
1. GRANULOMETRÍA  
2. CLASIFICACIÓN AASHTO  
3. CLASIFICACIÓN USC  
4. APAGAR HP  
5. SALIR"  
1 DISP  
0 WAIT  
→ i  
«  
CASE  
'i==92.1'  
THEN GRANULOMETRIA  
END  
'i==93.1'  
THEN AASHTO  
END  
'i==94.1'  
THEN USC  
END  
'i==82.1'  
THEN OFF  
END  
'i==83.1'  
THEN TEXT -40 SF END
```

```

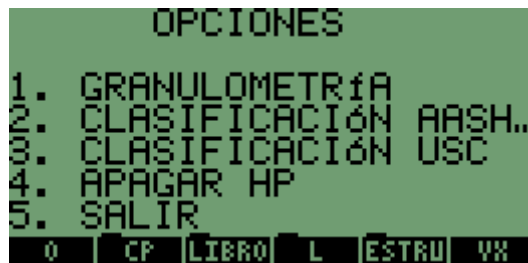
'i ≠ 92.1
OR
i ≠ 93.1
OR
i ≠ 94.1
OR
i ≠ 82.1
OR
i ≠ 83.1'
THEN INICIO
END
END

```

»

»

Grabamos Este código bajo el nombre INICIO y ejecutamos. Se vera así:



Si presionamos 2 por ejemplo:



Si presionamos 4:



Bueno, era de esperarse no?

COMENTARIOS:

1. Las variables GRANULOMETRÍA, AASHTO y USC son las variables que se ejecutarían inmediatamente después de presionar la tecla 1 2 o 3. En este caso llevan estos nombres, pero podría ser cualquier nombre.

2. Al teclear 2 no se ejecuto nada, solo se puso el nombre AASHTO en la pila. Si la variable AASHTO estuviera creada pues obviamente habría ejecutado dicha variable.

3. Nótese que los nombres GRANULOMETRÍA, AASHTO Y USC están sin los delimitadores ' '

También vemos que aparece un comando nuevo: **TEXT**. Este comando actualiza la pantalla, es decir "REFRESCA" la pantalla. Este comando es recomendado después del comando 0 WAIT, ES DECIR; DESPUÉS DE ESPERAR UNA TECLA.

10.3 ESTRUCTURAS DE BUCLE

Las Estructuras de bucle permiten que un programa ejecute una secuencia de comandos varias veces. Este numero de veces se especifica al inicio del programa.

10.3.1 START...NEXT

Para entender este tipo de estructura veamos el siguiente ejemplo: Escribamos un programa que ensamble las matrices de rigidez de los elementos de una cercha plana y los ponga en la pila, obviamente aplicando el método matricial:

La matriz de rigidez de un elemento de cercha plana en coordenadas globales esta dada por:

$$K = E \cdot A / L \begin{bmatrix} C^2 & S \cdot C & -C^2 & -S \cdot C \\ S \cdot C & S^2 & -S \cdot C & -S^2 \\ -C^2 & -S \cdot C & C^2 & SC \\ -S \cdot C & -S^2 & SC & S^2 \end{bmatrix}$$

Donde:

E = Modulo de elasticidad del material

A = Área de la sección transversal del elemento

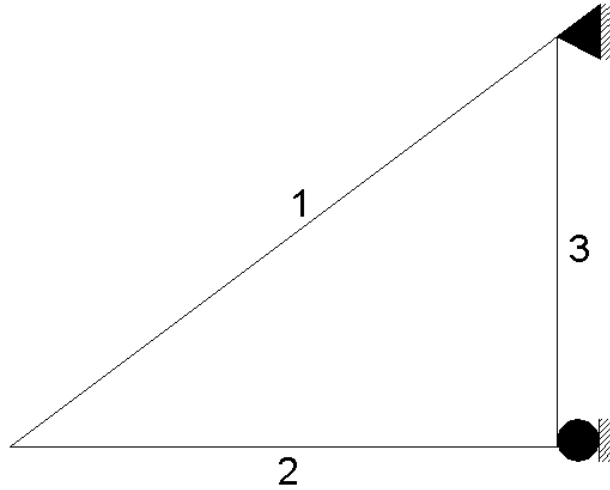
L = Longitud del elemento

C = Coseno del ángulo que describe el elemento con la horizontal

S = Seno del ángulo que describe el elemento con la horizontal.

Entonces los datos que necesita el programa para calcular la matriz de rigidez en coordenadas globales de cada elemento son E, A, L, θ

Veamos la forma y configuración de la cercha:



ELEMENTO 1:
 $L = 2.5 \text{ m}$
 $\theta = 36.87^\circ$
 $A = 0.00025 \text{ m}^2$
 $E = 200 * 10^6 \text{ KPA}$

ELEMENTO 2:
 $L = 2 \text{ m}$
 $\theta = 0^\circ$
 $A = 0.0004 \text{ m}^2$
 $E = 200 * 10^6 \text{ KPA}$

ELEMENTO 3:
 $L = 1.5 \text{ m}$
 $\theta = 90^\circ$
 $A = 0.00015 \text{ m}^2$
 $E = 200 * 10^6 \text{ KPA}$

Teniendo ya estos datos veamos el código:

« "Cuantos elementos son?"

" " INPUT OBJ→

1 SWAP

START

"Datos asi:

E A L θ " " "

INPUT OBJ→

→ E A L θ

«

E A * L / • COS 2 ^ *

E A * L / • SIN • COS **

E A * L / • COS 2 ^ NEG *

E A * L / • SIN • COS NEG **

E A * L / • SIN • COS **

E A * L / • SIN 2 ^ *

E A * L / • SIN • COS NEG **

E A * L / • SIN 2 ^ NEG *

E A * L / • COS 2 ^ NEG *

E A * L / • SIN • COS NEG **

E A * L / • SIN • COS **

E A * L / • SIN • COS NEG **

E A * L / • SIN 2 ^ NEG *

E A * L / • SIN • COS **

E A * L / • SIN 2 ^ *

{ 4 4 }

→ARRY

0 RND

»

NEXT

»

"Bueno, realmente NEXT incrementa el valor del contador por 1"
Al correr el programa vemos lo siguiente:

Entra el numero de elementos

Pone 1 en el n 2 y el # de elem en el n 1

Inicia el contador de bucle

Entra E A L θ

Define variable local para E A L θ y así poderlas llamar varias veces

Operaciones para cada elemento de la matriz

Dimensiones de la matriz

Forma la matriz

0 Posiciones decimales

Continúa con el segundo elem.

```
RAD XYZ HEX R= 'X' PRG
<HOME>
Cuantos elementos son?
3
0 CP LIBRO L ESTRU VX
```

Datos para el elemento 1:

```
RAD XYZ HEX R= 'X' PRG
<HOME>
Datos asi:
E A L θ
...E6 0.00025 2.5 36.87
0 CP LIBRO L ESTRU VX
```

Datos para el elemento 2:

```
RAD XYZ HEX R= 'X' PRG
[HOME]
-----
Datos asi:
E A L 0

20E6 0.0004 2 0
0 CP LIBRO L ESTRU VX
```

Datos para el elemento 3:

```
RAD XYZ HEX R= 'X' PRG
[HOME]
-----
Datos asi:
E A L 0

200E6 0.00015 1.5 90
0 CP LIBRO L ESTRU VX
```

Las respuestas se verán así en la pila:

```
DEG XYZ HEX R= 'X'
[HOME]
-----
2: [[ 40000. 0. -4000...
1: [[ 12800. 9600. -1...
   [[ 9600. 7200. -96...
   [[ -12800. -9600. ...
   [[ -9600. -7200. 9...
0 CP LIBRO L ESTRU VX
```

Es decir no se entiende nada.

COMENTARIOS:

1. La entrada de datos de cada elemento se puede optimizar mediante una plantilla de entrada, tema que se explicara mas adelante.
2. En este caso eran tres elementos, pero que pasaría si fueran 30?. Que después de ir en el cuarto o quinto elemento existe la posibilidad de que ya no sepamos en que elemento vamos. Esto se arregla escribiendo el código ya no con **STAR...NEXT** sino con **FOR NEXT**, el cual se explica a continuación.
3. Las respuestas son un poco difíciles de interpretar ya que simplemente son puestas en la pila. A continuación mediante **FOR NEXT** se optimizara muchísimo esto.

10.3.2 FOR...NEXT

Este tipo de estructura a diferencia de **START NEXT** nos permite utilizar el contador dentro de la cláusula-bucle

Para entender mejor este tipo de estructura escribamos el mismo ejemplo anterior pero ahora utilizando la estructura **FOR NEXT**.

En este ejemplo ya no dejaremos los resultados (matrices de rigidez de cada elemento) "regados" en la pila sino los grabaremos con el nombre ELEM*X* donde *X* es el numero del elemento:

Veamos entonces el código:

```
« "Cuantos elementos son?"
"" INPUT OBJ→
1 SWAP
FOR i                               Inicia la estructura
"Datos EL" i →STR +
" asi: E A L θ"
+
" " INPUT OBJ→
→ E A L θ
«
E A * L / θ COS 2 ^ *
E A * L / θ SIN * COS **
E A * L / θ COS 2 ^ NEG *
E A * L / θ SIN θ COS NEG **
E A * L / θ SIN θ COS **
E A * L / θ SIN 2 ^ *
E A * L / θ SIN θ COS NEG **
E A * L / θ SIN 2 ^ NEG *
E A * L / θ COS 2 ^ NEG *
E A * L / θ SIN θ COS NEG **
E A * L / θ COS 2 ^ *
E A * L / θ SIN θ COS **
E A * L / θ SIN θ COS NEG **
E A * L / θ SIN 2 ^ NEG *
E A * L / θ SIN θ COS **
E A * L / θ SIN 2 ^ *
{ 4 4 }
→ARRY
0 RND
"ELEM"
i →STR +
OBJ→
STO
»
NEXT
»
```

Ejecutemos este código:

```
RAD XYZ HEX R= 'X' PRG
{HOME}
-----
Cuantos elementos son?

3
W CP LIBRO L ESTRU VX
```

Ahora pedirá los datos del elemento 1 pero ahora si nos dice que elemento es (EL1):

```
RAD XYZ HEX R= 'X' PRG
{HOME}
-----
Datos EL1 asi: E A L 0

..E6 0.00025 2.5 36.87
W CP LIBRO L ESTRU VX
```

Ahora pedirá los datos del elemento 2 pero ahora si nos dice que elemento es (EL2):

```
RAD XYZ HEX R= 'X' PRG
{HOME}
-----
Datos EL2 asi: E A L 0

200E6 0.0004 2 0
ELEN1 W CP LIBRO L ESTRU
```

Ahora pedirá los datos del elemento 3 pero ahora si nos dice que elemento es (EL3):

```
RAD XYZ HEX R= 'X' PRG
{HOME}
-----
Datos EL3 asi: E A L 0

200E6 0.00015 1.5 90
ELEN2|ELEN1 W CP LIBRO L
```

Y tendremos los resultados grabados:

```
RAD XYZ HEX R= 'X'
<HOME>
5:
4:
3:
2:
1:
ELEM3|ELEM2|ELEM1| W | CP | LIBRO
```

↓ ↓ ↓
RESULTADOS

COMENTARIOS:

1. Observemos que el **FOR** va acompañado de **i**. Esta letra nos está representando el contador lo que quiere decir que **i tomara el valor del contador**. En este caso 1, 2 o 3. No necesariamente tiene que ser **i**, puede tomar cualquier carácter alfa
2. Por último recomiendo hacer un **DEBUG** a este código y si no quedo algo claro lo van a entender muy bien ya que así van a poder ver como trabaja la pila durante la ejecución del programa.

10.3.3 DO...UNTIL...END

Esta estructura ejecuta la secuencia cláusula-bucle de forma repetida hasta que se cumple una condición específica. Veamos un ejemplo:

Escribamos un programa que evalúe repetidamente la raíz cuadrada de un número puesto en el nivel 1 de la pila hasta que este número sea igual a 1. Es un ejemplo muy sencillo.

.....
Veamos el código:

```
« CLLCD
DO
√
DUP DUP
→STR 1 DISP
0.1 WAIT
UNTIL
1 ==
END
»
```

Ejecutemos este código:
Debemos poner un número en la pila. Ej; 9.999999999999E499

```
1.00005260354
W | CP | LIBRO | L | ESTRU | VX
```

COMENTARIOS:

1. Este código se entiende perfectamente haciendo **DEBUG**.

NOTA: - Inténtelo con un solo **DUP**.

- inténtelo sin el 0.1 WAIT

10.3.4 WHILE...REPEAT...END

Esta estructura trabaja de manera similar a **DO...UNTIL...END** con la diferencia de que primero ejecuta la condición y luego el bucle. Veamos el ejemplo anterior pero con **WHILE...REPEAT...END**:

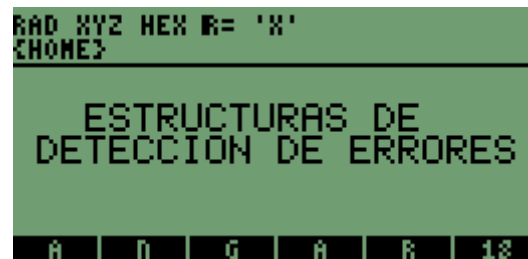
```
« CLLCD
WHILE DUP 1 ≠
REPEAT √ DUP
→STR 1 DISP
0.1 WAIT
END
»
```

COMENTARIOS:

1. Este código se entiende perfectamente haciendo **DEBUG**.

NOTA: - Inténtelo con un solo **DUP**.

inténtelo sin el 0.1 WAIT



11. ESTRUCTURAS DE DETECCION DE ERRORES

11. ESTRUCTURAS DE DETECCIÓN DE ERRORES.

11.1 DEFINICIÓN

Las estructuras de detección de errores permiten que los programas detecten o intercepten las situaciones de error las cuales provocarían la interrupción de la ejecución del programa.

11.2 IFERR...THEN...END

La estructura de IFERR...THEN...END es la siguiente:

```
« IFERR <acción sospechosa>  
THEN < acción a ejecutar si se produjo error>  
END  
»
```

Empecemos con un ejemplo muy sencillo:

Escribamos un programa que suma dos numero reales puestos en la pila. Una condición de error seria que hubiera un solo numero en la pila o que no fueran del mismo tipo de variable los argumentos puestos en la pila, Por lo tanto la acción sospecha seria +. Veamos este sencillo código:

```
« IFERR +  
THEN  
"Falta un dato"  
CLLCD MSGBOX  
END  
»
```

Al ejecutar con dos números en la pila:



Ejecutando con un solo numero en la pila (condición de error):



Se produce también un error cuando durante la ejecución de un programa se pulsa la tecla **ON**. Pruébenlo como ejercicio para un programa en donde exista una aplicación del comando **WAIT**.



12. VENTANA DE OPCIONES.

12. VENTANA DE OPCIONES.

12.1 DEFINICIÓN

Una ventana de opciones, como su nombre lo indica es una ventana que puede contener una o varias opciones, las cuales nos permiten seguir una ruta dentro de un programa.

12.2 EL COMANDO CHOOSE

Esta ventana de opciones se crea mediante el comando CHOOSE. Este comando toma tres argumentos de la pila para poder ejecutarse, estos argumentos son en orden los siguientes:

Un STRING el cual contiene el título de la ventana.

Una lista con las opciones que queremos ubicar en la ventana.

Un índice de posición.

12.3 FORMAS DE UTILIZACIÓN DEL COMANDO CHOOSE

El comando CHOOSE se puede aplicar de dos maneras, de hecho lo único que varía es la lista, es decir el segundo argumento que toma el comando para ejecutarse. Veamos un primer ejemplo:

EJEMPLO 1:

Escribamos un pequeño programa que cree una ventana con las siguientes opciones:

- Cercha
- Viga
- Pórtico

Veamos el código:

```
« CLLCD           Pon en blanco la pila
"ELIJA UNA OPCIÓN" Titulo de la ventana (STRING)
{ CERCHA
  VIGA
  PORTICO }      Opciones (lista)
1                Índice de posición
CHOOSE
»
```

Lo cual muestra lo siguiente:



Esto si esta activo el FLAG 90 el cual muestra el texto en FONT pequeño.



Veamos el código desactivando este FLAG

```
« -90 CF CLLCD  
"ELIJA UNA OPCION"  
{ CERCHA VIGA PORTICO }  
1  
CHOOSE  
»
```



NOTA: Cual es mejor? A gusto de cada uno.
Con las teclas que contienen las flechas nos podemos mover arriba o abajo.
Elijiendo la opción 3:



Presionando OK devolverá los siguiente a la pila:



COMENTARIOS:

Este código no nos permite poner texto que contenga mas de dos palabras, me explico; si fuera por ejemplo 'cercha plana' no podríamos ponerlo así. Se vería en una línea 'cercha' y en una segunda línea 'plana'

Veamos ahora una segunda forma de utilizar este comando **CHOOSE** la cuál nos permite poner mas de una palabra en una sola línea.

Considerando el mismo ejemplo anterior pero ahora las opciones serán:

- Cercha plana
- Viga continua
- Viga empotrada

Veamos el nuevo código:

```
« CLLCD  
"ELIJA UNA OPCIÓN"  
{  
  { "CERCHA PLANA" 1 }  
  { "VIGA CONTINUA" 2 }  
  { "VIGA EN EMPOTRADA" 3 }  
}  
1  
CHOOSE  
»
```

COMENTARIOS

1. Como se puede observar este código si nos permite poner mas de una palabra en una sola línea.
2. Notemos que las opciones a diferencia del código anterior ahora van dentro de "" (STRING)
3. Observemos también que la opción y su índice de posición ahora son una lista anidada dentro de otra lista que contiene todas las opciones.
4. Al ejecutar este código se ve algo como esto:



Eligiendo la opción 3

```
RAD XYZ HEX R= 'X'  
{HOME}  
-----  
0:  
4:  
00:  
2:  
1: 3  
1. 1.  
W CP LIBRO L ESTRU VX
```

El entero del nivel 2 de la pila es el índice de posición de la opción. Así por ejemplo hubiésemos elegido cercha plana se hubiera devuelto lo siguiente:

```
RAD XYZ HEX R= 'X'  
{HOME}  
-----  
0:  
4:  
00:  
2: 1  
1: 1.  
1. 1.  
W CP LIBRO L ESTRU VX
```

Dado el caso que no se presione la tecla de menú **OK** sino la tecla de menú **CANCL** se devolverá 0 a la pila

En resumen esta son las dos formas que toman los argumentos que necesita el comando **CHOOSE** para su ejecución. Veamos ahora un ejemplo completo utilizando este comando y aplicando la segunda forma:

Escribamos un programa que llamaremos **KTULU** con 3 opciones:

La primera nos permitirá averiguar la cantidad de memoria disponible de nuestra HP en KB (Kilobytes)

La segunda nos permitirá averiguar el numero de librerías instaladas en el puerto

La tercera nos da la opción salir del programa.

Vamos entonces al código:

« -90 SF IFERR		Si error
CLLCD		Pone la pantalla en blanco
" ELIJA UNA OPCION"		Titulo de la ventana
{ {"Memoria Disp." 1 }	}	
{ "Puerto 0" 2 }		
{ "Salir" 3 }		Lista con las opciones
}		
1		Índice de posición
CHOOSE		Ejecuta el comando CHOOSE
DROP		Elimina el 1 dejado en la pila
→ i		Define var. local para 1 2 o 3
« CASE ' i==1'	En caso de que i igual a 1 (es decir memoria	disponible)
THEN	Entonces	
CLLCD	Pone la pantalla en blanco	
-40 CF	Quita el reloj de la pantalla	
MEM	Pone la cantidad de memoria disp. En la pila	
1000 /	Divide por 1000 para pasar a KB	
0 RND	0 lugares decimales	
→STR	Convierte a STRING	
" KB" +	Añade KB al resultado de mem disponible	
"Memoria disponible:"	Pone el texto "Memoria disponible" en la pila	
1 DISP	Pone el texto anterior en la pantalla	
2 DISP	Pone el texto de MEM disponible en pantalla	
0 WAIT	Espera una tecla	
DROP	Borra la semilla dejada por la tecla pulsada	
END	Termina	
' i==2'	En caso de que i igual a 2 (es decir puerto 0)	
THEN	Entonces	
0 PVARs	Comando que nos devuelve una lista con la cantidad	
	de librerías en puerto 0 representadas por su numero (nivel	
	2) y la memoria disponible (nivel 1)	
	Borra el nivel 1 de la pila ya que no necesitamos el dato	
	de cantidad de mem disponible.	
	Obtiene el numero (cantidad) de elementos de la lista	
	Lo convierte a STRING	
	Pone el texto LIBS y lo añade a # de LIBS	
	Pone el texto "LIBS EN PUERTO 0:"	
	Pone el texto anterior en la pantalla	
	Pone la cantidad (numero) de LIBS en la pila.	
	Espera una tecla	
	Borra la semilla dejada por la tecla presionada	
	Termina	
	En caso de que i igual a (es decir salir)	
	Entonces	
	Termina	
	Sale del CASE	
»		
THEN	Si se produjo error entonces	
CLLCD "Se produjo un error"		
MSGBOX	Muestra "Se produjo un error" mediante MSGBOX	
KTULU	Vuelve a ejecutar el programa	
END	Termina	
»		

Grabamos bajo el nombre **KTULU** y ejecutamos. Obsérvese que se **modifico** el FLAG 90:



Si elegimos la opción 1



Si elegimos la opción 2



Si presionamos CANCL



COMENTARIOS:

1. Vemos un comando nuevo llamado **PVARS** que nos devuelve una lista con la cantidad de librerías en puerto 0 representadas por su numero (nivel 2) y la memoria disponible (nivel 1).

2. El comando **CHOOSE** es bastante eficiente cuando tenemos muchas opciones dentro de nuestras aplicaciones.

EJEMPLO 2:

Ahora supongamos que lo que tenemos es una lista con muchos nombres (como en una agenda) y lo que queremos es seleccionar dicho nombre:

Veamos la lista con los nombres:

```
{  
"Andres Garcia"  
"Santiago Jose Lopez"  
"Fabian Herrera"  
"Carlos Diaz"  
"Cristina Santamaria"  
"Joe satriani"
```

```

"Juan Gallardo"
"Armando De castro"
"Carlos de Castro"
"Elkin Cardenas"
}

```

Grabamos bajo el Nombre NOMBRES:

```

RAD XYZ HEX R= 'X'
[HOME]
5:
4:
3:
2:
1:
NOMBRE CP LIBRO L ESTRU YX

```

NOMBRES

Veamos el codigo:

```

« " CONTACTOS"
NOMBRES
1
CHOOSE
DROP
»

```

Grabamos bajo el nombre AGENDA y ejecutamos:

```

RAD CONTACTOS
[CHOM]
5: Andres Garcia
4: Santiago Jose Lopez
3: Fabian Herrera
2: Carlos Diaz
1: Cristina Santamaria
   Joe satriani
CANCL OK

```

Si elegimos a Santiago Jose Lopez:

```

RAD XYZ HEX R= 'X'
[HOME]
5:
4:
3:
2:
1: "Santiago Jose Lop...
AGEND NOMBRE CP LIBRO L ESTRU

```

Este código se relaciona directamente con el código de la pagina 58

```
RAD XYZ HEX R= 'X'  
{HOME}  
PLANTILLAS DE ENTRADA  
A D G A R 18
```

13. PLANTILLAS DE ENTRADA.

13. PLANTILLAS DE ENTRADA.

13.1 DEFINICIÓN

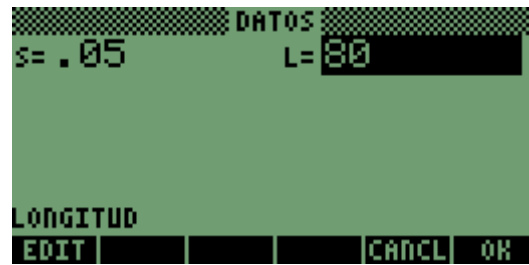
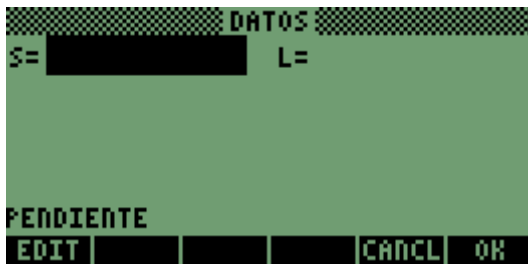
Las plantillas de entrada son básicamente una forma de entrar datos a un programa de manera bastante cómoda y eficiente ya que nos permite hacer algunas cosas mas que por ejemplo una entrada mediante el comando **INPUT**.

La plantillas de entrada se ejecutan o se crean mediante el comando **INFORM** el cual necesita tres argumentos básicos para su ejecución que son:

- Un titulo (STRING)
- Un conjunto de campos (Una lista)
- Una línea de mensajes (Un STRING).

Claro esta que una plantilla de entrada también nos sirve para mostrar resultados arrojados por un programa como se vera mas adelante.

Un ejemplo sencillo de una plantilla de entrada sería el siguiente:



13.1 FORMAS DE UTILIZAR LAS PLANTILLAS DE ENTRADA

Existen dos formas básicas de utilizar las plantillas de entrada:

- Una en donde no especificamos el argumento permitido **sin línea de mensajes**.
- Otra en donde podemos especificar el argumento, el numero de columnas y además ponemos una línea de mensajes.

Veamos entonces la primera forma:

- SIN ESPECIFICAR EL ARGUMENTO A ENTRAR Y SIN LÍNEA DE MENSAJES:

Para esto veamos el código del ejemplo anterior (el de pendiente y longitud):

```
«
"ENTRADA DE DATOS"           Titulo
{
{"Q="} _____
{"A="} _____
}
}
DUP DUP                      Lista
INFORM                       Duplica la lista
                              Crea la plantilla de entrada
»
```

Al ejecutar este código veremos lo siguiente:



COMENTARIOS:

1. Observemos que en la parte baja de la pantalla no hay un mensaje de ayuda, es decir; no sabemos que quiere decir S ni tampoco L.
- Hay solo una columna pero que tal si queremos dos?, es decir; que S no quede sobre L?
3. En esta plantilla no esta especificado el tipo de argumento que el usuario debe introducir, por ende se puede introducir cualquier argumento, veamos algunos



PROGRAMAS



SIMBÓLICOS



LISTAS

Pero que tal si la plantilla que creamos es para un programa que utiliza solo argumentos numéricos?. Ahora veamos la segunda forma básica de una plantilla de entrada:

-ESPECIFICANDO EL ARGUMENTO, EL NUMERO DE COLUMNAS Y UNA LÍNEA DE MENSAJES O STRING DE AYUDA:

Revisemos el ejemplo anterior pero ahora especificando el argumento, el numero de columnas y una línea de mensajes o STRING de ayuda:

«
"ENTRADA DE DATOS"

```

{
{"S=" "PENDIENTE" 0}
{"L=" "LONGITUD" 0}
}
{ 2 0 }
{ } DUP
INFORM
»

```

Ejecutándolo:



COMENTARIOS:

1. Intente introducir un argumento que no sea numérico.
2. Cree la plantilla cambiando a { 1 0 } (Lista que contiene el numero de columnas y el numero de espacios entre "Q" y el espacio para poner el valor) por { 2 0 }.

Ya teniendo las herramientas para introducir datos en programa mediante una plantilla de entrada entonces escribamos el siguiente código que nos permite ensamblar las matrices de rigidez de cada elemento de una cercha en coordenadas globales, entrando los datos necesarios mediante plantillas de entrada.

La matriz de rigidez de un elemento de cercha plana en coordenadas globales esta dada por:

$$[K] = E \cdot A / L \begin{bmatrix} C^2 & S \cdot C & -C^2 & -S \cdot C \\ S \cdot C & S^2 & -S \cdot C & -S^2 \\ -C^2 & -S \cdot C & C^2 & SC \\ -S \cdot C & -S^2 & SC & S^2 \end{bmatrix}$$

Donde:

E = Modulo de elasticidad del material

A = Área de la sección transversal del elemento

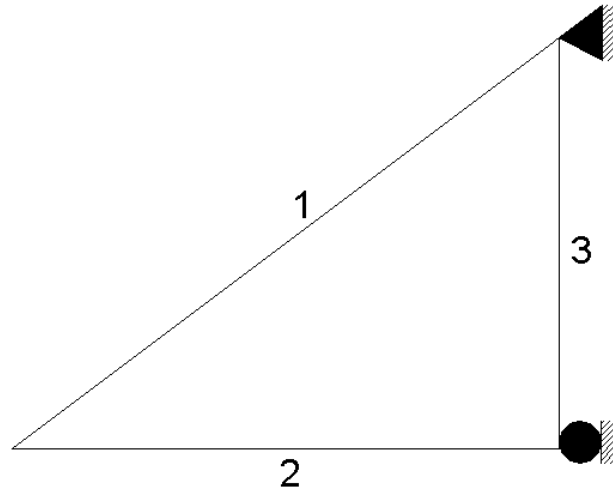
L = Longitud del elemento

C = Coseno del ángulo que describe el elemento con la horizontal

S = Seno del ángulo que describe el elemento con la horizontal.

Entonces los datos que necesita el programa para calcular la matriz de rigidez en coordenadas globales de cada elemento son E , A , L , θ

Veamos la forma y configuración de la cercha:



ELEMENTO 1:
 $L = 2.5 \text{ m}$
 $\theta = 36.87^\circ$
 $A = 0.00025 \text{ m}^2$
 $E = 200 * 10^6 \text{ KPA}$

ELEMENTO 2:
 $L = 2 \text{ m}$
 $\theta = 0^\circ$
 $A = 0.0004 \text{ m}^2$
 $E = 200 * 10^6 \text{ KPA}$

ELEMENTO 3:
 $L = 1.5 \text{ m}$
 $\theta = 90^\circ$
 $A = 0.00015 \text{ m}^2$
 $E = 200 * 10^6 \text{ KPA}$

Teniendo ya estos datos veamos el código:

```
« "NUMERO DE ELEMENTOS"  
{  
{"NO:" "NUMERO DE ELEMENTOS" 0}  
}  
{1 0}  
{ } DUP  
INFORM  
DROP OBJ→  
DROP  
1 SWAP
```

```

FOR i
"DATOS ELEMENTO " i
→STR +
{
{"E" "MODULO DE ELASTICIDAD EN KPA" 0}
{"A" "AREA DE LA SECCION EN M^2" 0 }
{"L" "LONGITUD DEL ELEMENTO EN M" 0}
{"θ" "ANGULO CON LA HORIZONTAL" 0}
}
{2 0}
{ } DUP
INFORM
DROP
OBJ→
DROP
CLLCD
"Ensamblando matriz
elemento " i →STR +

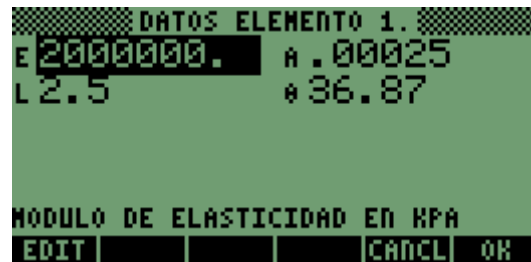
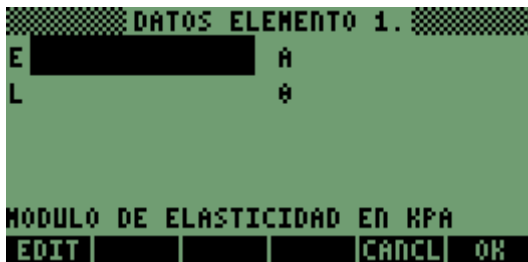
1 DISP
"Espere por favor. . "
4 DISP
→ E A L θ
«
E A * L / θ COS 2 ^ *
E A * L / θ SIN • COS **
E A * L / θ COS 2 ^ NEG *
E A * L / θ SIN • COS NEG **
E A * L / θ SIN • COS **
E A * L / θ SIN 2 ^ *
E A * L / θ SIN • COS NEG **
E A * L / θ SIN 2 ^ NEG *
E A * L / θ COS 2 ^ NEG *
E A * L / θ SIN • COS NEG **
E A * L / θ COS 2 ^ *
E A * L / θ SIN • COS **
E A * L / θ SIN • COS NEG **
E A * L / θ SIN 2 ^ NEG *
E A * L / θ SIN • COS **
E A * L / θ SIN 2 ^ *
{ 4 4 }
→ARRY
0 RND
»
"ELE" i →STR
+ OBJ→
STO TEXT
NEXT
»

```

Al ejecutar el código veremos lo siguiente:



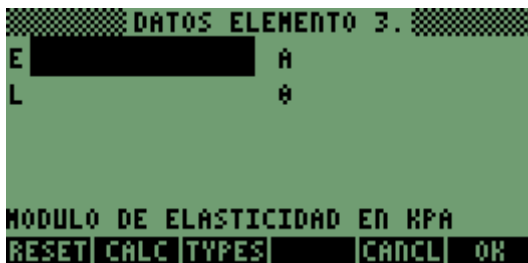
Presionamos OK y nos pedirá datos elemento por elemento:



Presionamos OK y nos pedirá los datos del elemento 2:



Presionamos OK y nos pedirá los datos del elemento .3:



Presionamos OK.

Los resultados quedaran grabados en el directorio actual bajo los nombres E1, E2, E3.

```
RAD XYZ HEX R= 'X'  
{HOME}  
5:  
4:  
3:  
2:  
1:  
ELE3. ELE2. ELE1. AGEND NOMBRA CP
```



COMENTARIOS:

1. Primero que todo recomiendo hacer un **DEBUG** a este código por si no se entendió algo.
2. Intenten escribir este código pero no ensamblando la matriz de cada elemento a medida que se introducen los datos, sino que ensamble todas las matrices al final.
3. Importante tener la HP en modo DEG, de lo contrario los resultados no serán correctos. Esto se logra poniendo la palabra **DEG** al principio del Programa.

```
RAD XYZ HEX R= 'X'  
CHOME3  
-----  
EL ENTORNO SOLVR  
-----  
A | N | G | A | R | 18
```

14. EL ENTORNO SOLVR.

14. EL ENTORNO SOLVR.

14.1 COMO SOLUCIONAR ECUACIONES DENTRO DE UN PROGRAMA.

Como definición previa, el Entorno **SOLVR** es una herramienta de la HP que nos brinda la posibilidad de solucionar ecuaciones en las cuales la variable o incógnita es difícil de despejar, es decir; Aquellas ecuaciones donde tenemos que hacer "tanteo" para encontrar el valor de la incógnita. El entorno SOLVR utiliza un conjunto de por lo menos dos ecuaciones que van dentro de una lista

Por ejemplo para encontrar el factor de fricción en tuberías que conducen algún tipo de fluido a presión se utiliza la Ecuación de COLEBROOK que viene dada como:

$$1/\sqrt{f} = - (2 * \text{LOG} (1 / 3.7 * K_s / D + 2.51 / R * \sqrt{f}))$$

Donde:

f = Factor de fricción (Adimensional)

Ks = Coeficiente de rugosidad del material de la tubería (en mm)

D = Diámetro de la tubería. (En mm)

R = # de Reynolds. (Adimensional)

Como podemos ver, en esta ecuación es bastante complicado despejar la incógnita f, tendríamos que hacer tanteos hasta que se cumpla la igualdad. Pero entonces veamos como se utiliza el entorno SOLVR para solucionar este tipo de ecuaciones, considerando el siguiente ejemplo:

EJEMPLO 1:

Se quiere encontrar el valor del factor fricción en una tubería que conduce agua con los siguientes datos:

Ks = 0.12 mm (Tubería de concreto CCP)

D = 300 mm

R = 421847.07

Veamos el código para luego explicarlo:

```
« -3 CF
"Ks (En mm)" " "
INPUT OBJ→
"DIAMETRO (En mm):" " "
INPUT OBJ→
"# DE REYNOLDS:" " "
INPUT OBJ→
→ Ks D R
« CLLCD
"Un momento por favor
Resolviendo ecuacion.."
3 DISP
' 1/fF=-(2*LOG(1/3.7*(Ks/D)+2.51/(R*fF))) ' EVAL
' Y=0 '
2 →LIST
'EQ' STO
MINIT
F MCALC
F MROOT
5 RND
→STR
"f=" SWAP +
MSGBOX
»
»
```

Ejecutemos el código:

```
RAD XYZ HEX R~ 'X' PRG
{HOME}
Ks (En mm)

0.12
AGEND|NOMBR| CP | LIBRO | L | ESTRU
```

```
RAD XYZ HEX R~ 'X' PRG
{HOME}
DIAMETRO (En mm):

300
AGEND|NOMBR| CP | LIBRO | L | ESTRU
```

```
RAD XYZ HEX R~ 'X' PRG
{HOME}
# DE REYNOLDS:

421847.07
AGEND|NOMBR| CP | LIBRO | L | ESTRU
```

```
Un momento por favor=
Resolviendo ecuacion..

AGEND|NOMBR| CP | LIBRO | L | ESTRU
```

```
Un Res f=.01719 r=
n..

OK
```

COMENTARIOS:

1. En la línea 1 vemos -3 **CF** lo que nos indica que estamos cambiando el FLAG 3. Esto es de vital importancia ya que este FLAG controla el modo de función numérica o simbólica. Siempre debe estar en simbólica, para esto ponemos -3 **CF**.
2. Luego vemos que pide la entrada de los datos necesarios mediante el comando **INPUT**.
3. → **Ks D R** Define las variables locales Ks, D, R
4. **CLLCD** Pone la pantalla en blanco.
5. "Un momento por favor Resolviendo ecuación.." 3 **DISP**. Pone el texto en el nivel 3 de la pantalla.
6. ' $1/fF=-(2*\text{LOG}(1/3.7*(Ks/D)+2.51/(R*fF)))$ ' **EVAL**. Realiza la operación.
7. ' **Y=0** ' Conformamos la ecuación 'Y=0' Recordemos que el entorno SOLVR utiliza dos o más ecuaciones que van dentro de una lista. Esta segunda ecuación se puede conformar como una quiera siempre y cuando no contenga variables que contenga la primera ecuación **en este caso**.

8. 2 →**LIST** Conformar la lista

9. 'EQ' **STO** Graba la lista con el nombre EQ

10. **MINIT**. Este comando es nuevo para nosotros, y lo que hace es definir la lista con las dos ecuaciones.

11. F **MCALC** Este comando también es nuevo para nosotros, y lo que hace es definir a F como la incógnita a despejar.

12. F **MROOT** Este comando también es nuevo para nosotros, y lo que hace es definir a F como la variable para la cual va a encontrar las raíces o ceros de la ecuación.

13. 5 **RND**. Trunca la respuesta a 5 lugares decimales.

14. →**STR** Convierte la respuesta a STRING para mostrarla mediante el comando **MSGBOX**.

15. "f=" **SWAP** +. "Etiqueta la respuesta" "f = 0.017"

16. **MSGBOX** Crea una ventana de dialogo para mostrar la respuesta.

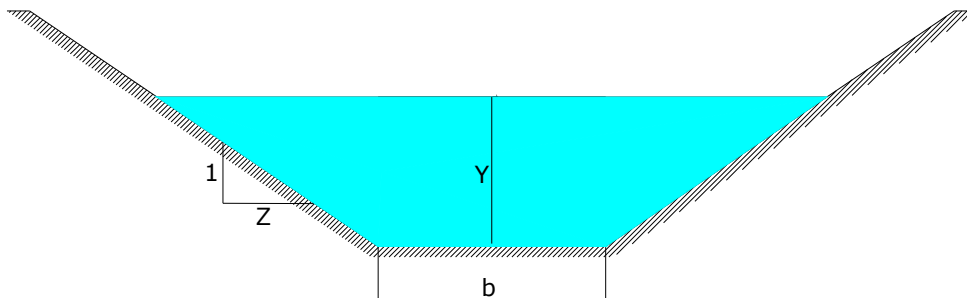
17. El programa nos deja grabadas algunas variables en el directorio actual como (F, MPAR, EQ) **Las cuales hay que eliminar.**

17. Recomiendo hacer un **DEBUG** a este código.

NOTA: Aquí en la 49G es necesario modificar el FLAG 105. Como ejercicio averigüen porque.

EJEMPLO 2:

Se desea calcular la altura de lamina (Y) en un canal trapezoidal que trabaja a flujo libre. La sección es como se indica:



La ecuación para determinar la altura de lamina Y viene dada por la ecuación:

$$[(b + Z * Y) * Y]^{2.5} / (b + 2 * Y * \sqrt{Z^2 + 1}) = (Q * n / S^{1/2})^{1.5}$$

Donde:

Y = Altura de la lamina

b = Base del canal en m

Z = Pendiente del talud

Q = Caudal en m³ / s

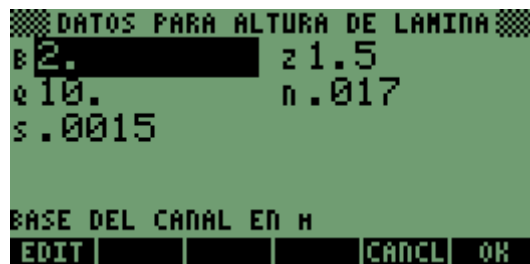
n = Coeficiente de fricción de Manning

S = Pendiente longitudinal
 Entonces suponiendo los siguientes datos:
 b = 2 m
 Z = 1.5
 Q = 10 m³ / s
 n = 0.017
 S = 0.0015
 Entonces veamos el código:

```

« "DATOS PARA ALTURA DE LAMINA"
{
{"B" "BASE DEL CANAL EN m" 0 }
{"Z" "PENDIENTE DEL TALUD" 0 }
{"Q" "CAUDAL EN M^3/S" 0 }
{"N" "COEFICIENTE DE MANNING" 0 }
{"S" "PENDIENTE LONGITUDINAL DEL CANAL" 0 }
}
{ 2 0 }
{ } DUP
INFORM
DROP
OBJ→ DROP
→ b Z Q n S
« CLLCD
'((b+Z*Y)*Y)^2.5/(b+2*Y*f(Z^2+1))=(Q*n/S^.5)^1.5'
EVAL
'X=0'
2 →LIST
'EQ' STO
MINIT
Y MCALC
Y MROOT
2 RND
"ALTURA DE LAMINA"
{{ "Y=" "ALTURA DE LAMINA EN M" 0 }}
{ 1 0 }
4 ROLL
1 → LIST
DUP
INFORM
»
»
  
```

Ejecutemos entonces este código:





COMENTARIOS:

1. Como se puede ver, aquí se utilizó una plantilla de entrada para mostrar un resultado.
2. Si algo no se entendió, recomiendo hacer un **DEBUG**.

Espero se haya entendido la forma como se utiliza el entorno **SOLVR** solucionador de ecuaciones, herramienta supremamente útil y fácil de usar.



15. EL ENTORNO PICT.

15. EL ENTORNO PICT.

15.1 DEFINICIÓN

El entorno PICT es una herramienta de la HP que nos permite crear y editar dibujos creados por nosotros mismos (que también llamaremos GROBS) y representaciones gráficas procedentes de datos matemáticos. Al igual que los demás objetos de la HP, los objetos gráficos pueden colocarse en la pila y archivarse dentro de variables.

En la pila un objeto gráfico se ve de la siguiente forma:

```
RAD XYZ HEX R~ 'X'  
<HOME>  
5:  
4:  
3:  
2:  
1: Graphic 131 x 64  
AGEND|NOMBR|CP|LIBRO|L|ESTRU
```

Donde:

131 es el ancho en puntos (píxeles)

64 es la altura en puntos (Píxeles)

El tamaño por defecto del entorno en coordenadas es (-6.5 a 6.5) en X y (-3.1 a 3.1) en Y.

15.2 FORMAS DE VISUALIZAR EL ENTORNO PICT

Existen dos formas básicas (entre otras) de visualizar el entorno PICT:

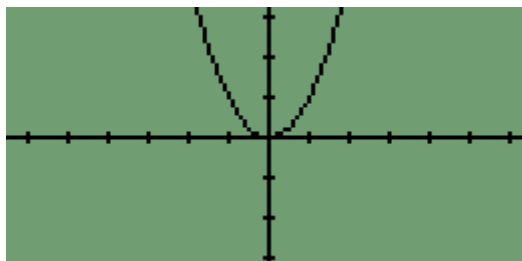
Veamos a través un código la primera forma:

```
« { } PVIEW
```

```
»
```

Este pequeño programa nos permite visualizar el entorno PICT bloqueando el teclado, es decir a la única tecla que responde es a ON.

Veamos que pasa al ejecutarlo:



Nos está mostrando el entorno PICT con el último GROB que se almacena allí sea proveniente de algún dato matemático como en este caso que lo último que se graficó fue la ecuación $Y=X^2$ o sea proveniente de alguna GROB personalizado que se visualizó en este entorno.

Sea cual sea la manera de visualizar el entorno, para que nos se vea lo último que visualizó allí utilizamos el comando **ERASE**. Veamos el código anterior con este nuevo comando que lo que hace es borrar por completo lo que hay en **PICT**.



Si presionamos **CANCL** volveremos a la pila:

```
RAD XYZ HEX R~ 'X'  
{HOME}  
5:  
4:  
3:  
2:  
1:  
AGEND|NOMBR|CP|LIBRO|L|ESTRU
```

Veamos a través un código la segunda forma:

```
« ERASE {#0 #0 }  
PVIEW  
0 WAIT  
»  
COMENTARIOS:
```

1. **{#0 #0 }** son las coordenadas por defecto de las esquinas del entorno **PICT**.
2. **PVIEW** nos muestra el entorno **PICT**. Pero debemos poner el **0 WAIT** para lo mantenga, de lo contrario se visualizaría muy rápido y luego volvería a verse la pila.
Veamos lo que hace el código:



3. Pruébenlo sin el **0 WAIT**.

4. Cuando visualizamos el entorno de esta forma se nos crea una variable llamada **PPAR** la cual contiene algunos parámetros del entorno como las dimensiones el tipo de función a graficar activada actualmente y el origen de coordenadas.

Estas son las formas básicas de visualizar el entorno **PICT**, que junto con otros comando que veremos a continuación nos permiten crear programas bastante elaborados.

15.3 COMO PONER Y VISUALIZAR TEXTO EN EL ENTORNO PICT.

15.3.1 MEDIANTE EL COMANDO REPL:

El comando REPL permite colocar texto en entorno utilizando tres argumentos a seguir:

- PICT

- Las coordenadas donde deseamos colocar el texto.

- Y un texto convertido en GROB. Para convertir el texto a GROB simplemente lo colocamos en la Pila dentro de delimitadores de STRING, le damos un tamaño que puede estar entre 0 y 3 y ejecutamos el comando **→GROB**. El cual lo que hace es convertir el STRING con el texto a **GROB**.

Veamos en ejemplo:

Deseamos colocar nuestro nombre en el entorno PICT en las coordenadas $-6.5, 3.1$ (esquina superior izquierda).

Veamos el código:

```
« ERASE {#0 #0 }
```

```
PVIEW
```

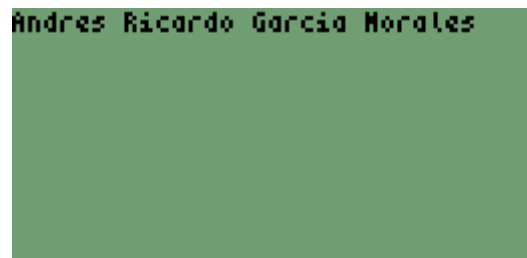
```
PICT (-6.5,3.1) "Andres Ricardo Garcia Morales"
```

```
1 →GROB REPL
```

```
0 WAIT
```

```
»
```

Ejecutemos el código:



Presionamos cualquier tecla (en este caso **ENTER**)




VARIABLE PPAR

COMENTARIOS:

1. Vemos Que nos creó la variable PPAR
2. Nos dejó el número de asignación de tecla de la última tecla pulsada.
3. En este caso se escogió el tamaño de letra más pequeño **1**
Veamos el código pero ahora borrando PPAR, borrando el número de asignación de tecla dejado y con tamaño de letra igual a **2**

```
« ERASE {#0 #0 }  
PVIEW  
PICT (-6.5,3.1) "Andres Garcia"  
2 →GROB REPL  
0 WAIT  
DROP  
{ PPAR } PURGE »
```



Pulsamos ENTER:



15.3.2 MEDIANTE EL COMANDO GXOR:

Este comando pone texto de la misma manera que en el ejemplo anterior, a diferencia de que si el fondo fuera negro pone la letra en blanco. Veamos un ejemplo:

Supongamos que en el entorno **PICT** ya está creado el siguiente GROB:



Y queremos colocar texto dentro del recuadro negro. Veamos como sería el código:

```
« {#0 #0 }  
PVIEW  
PICT (-4.5,0) "Andres Garcia"  
2 →GROB  
GXOR  
0 WAIT  
DROP  
{ PPAR } PURGE  
»
```

Se vería algo así:



Bueno, no me quedo muy en el centro que digamos pero el ejemplo sirve para ver la diferencia. Donde esta en blanco pone píxeles Negros (prende píxeles) y donde esta en negro pone píxeles blancos (apaga píxeles).

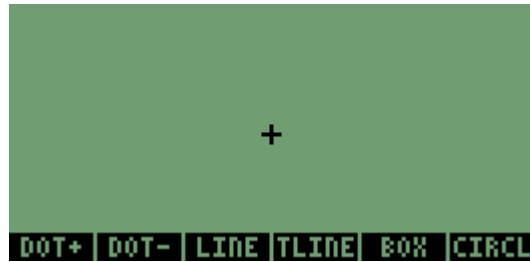
También podemos visualizar GROBS creados con anterioridad por nosotros mismos. Veamos un ejemplo en donde creamos el cuadro negro, lo ponemos en el entorno y le colocamos texto adentro a manera de presentación de un programa. Vamos a verlo de dos maneras:

- CREANDO EL GROB (cuadro negro en este caso) PRIMERO Y LUEGO COLOCÁNDOLO DENTRO DEL PROGRAMA:

Para crear el cuadro negro entramos al entorno PICT pulsando la tecla que contiene la flecha izquierda (segunda fila cuarta tecla):



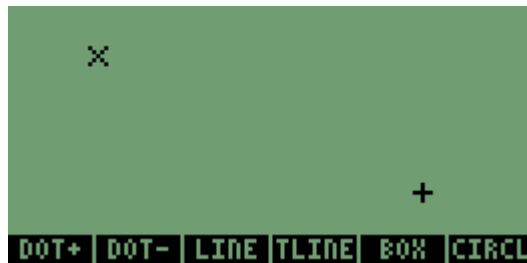
Pulsamos la tecla EDIT y en las teclas de MENÚ veremos las herramientas de dibujo



Para crear el cuadro negro lo haremos mediante la herramienta **BOX** la cual dibuja un recuadro así: Nos paramos con la cruz en donde queremos que quede la esquina superior del cuadro (en este caso el cuadro negro)



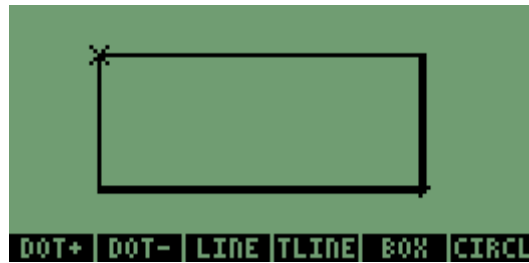
Ejecutamos el comando **BOX** y nos paramos con la cruz en donde queremos que quede la esquina inferior derecha del cuadro (en este caso cuadro negro)



Presionamos **BOX**:



Subimos un píxel hacia arriba y uno hacia la izquierda y volvemos a ejecutar **BOX**:



Repetimos esta operación (subir un píxel y uno hacia la izquierda) hasta que el cuadro este totalmente relleno:



NOTA: Para que la cruz se vea en blanco pulsamos la tecla +/-
Salimos del editor pulsando CANCL:

Volvemos al entorno pulsando la tecla que tiene la flecha a la izquierda:



Ubicamos la cruz en la esquina superior izquierda del cuadro negro:



Pulsamos la tecla **EDIT** luego la tecla **NEXT** dos veces (es decir **NEXT NEXT**) y



luego la tecla **SU**

Movemos la cruz a la esquina inferior derecha y de nuevo presionamos **SUB**



Salimos del entorno PICT presionando CANCL y ya tendremos nuestro GROB (cuadro negro) creado en la pila:

```
RAD XYZ DEC R~ 'X'  
{HOME}  
5:  
4:  
3:  
2:  
1:      Graphic 84 x 37  
OBJ+  +ARRAY +LIST +STR +TAG +UNIT
```

Lo grabamos bajo el nombre GRAF:

```
RAD XYZ DEC R~ 'X'  
{HOME}  
5:  
4:  
3:  
2:  
1:  
GRAF  PPAR  AGEND  NOMBR  CP  LIBRO
```



GRAF

Ahora veamos las coordenadas donde mas o menos quedaría centrado el texto que vamos a poner (en este caso el titulo del programa **TOPORAD**), teniendo en cuenta que va a ser en tamaño 2: Entramos el entorno presionando la tecla que contiene la flecha hacia la izquierda:



```
(X, Y)  EDIT CANCL
```

Y nos paramos donde mas o menos quedaría centrado el texto. Para obtener esas coordenadas presionamos la tecla ENTER y salimos del entorno presionando CANCL:

```

RAD XYZ DEC R~ 'X'
(HOME)
5:
4:
3:
2:
1: (-3.2, 1.3)
GRAF PPAR AGEND NOMBRE CP LIBRO

```

Para averiguar la coordenadas de la esquina superior izquierda donde vamos a colocar el cuadro lo hacemos de la misma forma. En este caso son (-4.4,2.1)

Ya sabemos cuales serian las coordenadas del cuadro negro, del texto que va dentro del cuadro negro y también tenemos nuestro GROB (cuadro negro) ya creado entonces vamos al código:

```

« ERASE {#0 #0 }
PVIEW
PICT
(-4.4,2.1)
GRAF
REPL
PICT
(-3.2,1.3)
"TOPORAD"
2
→GROB
GXOR
0 WAIT DROP
{ PPAR }
PURGE »

```

Ejecutemos Este código:



Como podemos ver que el texto no quedo tan centrado entonces modifiquemos sus coordenadas:

```

« ERASE {#0 #0 }
PVIEW
PICT
(-4.4,2.1)
GRAF
REPL
PICT

```

(-2.5,1.3)

"TOPORAD"

2

→GROB

GXOR

0 WAIT DROP

{ PPAR }

PURGE

»



Agreguémosle el autor y el año:



COMENTARIOS:

1. De la misma manera como se creo el cuadro negro se pueden crear diferentes dibujos como círculos triángulos arcos etc.
2. Para mucha gente es engorroso crear los **GROBS** utilizando el entorno **PICT** para lo cual existen otros programas para PC como **PAINT**, luego se utiliza un programa llamado **XnVIEW** el cual le coloca extensión **.grob** para que la HP lo pueda mostrar.

- CREANDO EL GROB DENTRO DEL PROGRAMA:

Veamos el código, mucha atención:

```
« ERASE {#0 #0 }
PVIEW
(-4.4,2.1)
'X' STO
(4.4,2.1)
'Y' STO
1 35
FOR i
X
DUP (0,-0.1) +
'X' STO
Y DUP (0,-0.1) +
'Y' STO
LINE
NEXT
PICT
(-2.5,1.3)
"TOPORAD"
2
→GROB
GXOR
PICT
(-4,0.3)
"Por: Santiago Lopez"
1
→GROB
GXOR
PICT
(-1,-0.5)
"2011"
1
→GROB
GXOR
0 WAIT DROP
{ X Y PPAR }
PURGE
»
```

Ejecutemos este código:



COMENTARIOS:

1. Vemos que aparece un comando nuevo llamado **LINE**. Este nos permite dibujar líneas en el entorno **PICT** dados dos pares de coordenadas (x1,y1) (x2,y2) con x2 >x1.
2. Recomiendo enfáticamente hacer un **DEBUG** a este código.
3. De la manera como se creo el cuadro negro se puede crear cualquier dibujo que queramos.



15.4 COMO PONER Y VISUALIZAR GRÁFICOS EN EL ENTORNO PICT

Bueno, parte de esto se explico en el numeral anterior pero no importa; veamos otros ejemplos:

EJEMPLO 1:

Vamos a suponer que queremos poner nuestro nombre en el entorno PICT pero ahora haciéndolo letra a letra, primero con una pausa (lapso de tiempo) entre letras y después sin pausa. Veamos el código:

- CON PAUSA:

```
« ERASE { #0 #0 }
PVIEW
(-6.5,3.1) 'CO' STO
"ANDRES R GARCIA M"
→ n
« 1 17 FOR i
PICT CO DUP
(0.4,0) +
'CO'
STO
n
i DUP
SUB
→STR 1
→GROB REPL
0.1 WAIT
NEXT
0 WAIT
{ CO PPAR }
PURGE » »
```

Ejecutemos este código:

```
A
```

```
AN
```

```
AND
```

```
ANDR
```

Así sucesivamente hasta formar el nombre completo:

```
ANDRES R GARCIA M
```

- SIN PAUSA:

Simplemente quitamos el WAIT y el texto se ubicara más rápido en pantalla.

EJEMPLO 2:

Ahora vamos suponer que lo que queremos graficar son datos estadísticos, en este caso una nube de puntos provenientes de tomar datos de 2 variables por ejemplo: altura (en msnm) vs temperatura (en °C). Los datos serian los siguientes (estos son arbitrarios a manera de ejemplo)

ALTURA (msnm)	TEMPERATURA C
0	38
300	34
600	32
900	31
1200	30
1500	26
1800	25
2100	20
2400	16
2700	13
3000	10
3300	8
3600	7
3900	5
4200	3
4500	1

Veamos el código completo de un programa que grafica estos datos a manera de puntos en el entorno **PICT**:

```

« "DATOS"
{ { "h=" "Alturas en msnm en {" }" 5. }
{ "T=" "Temperaturas en C en {" }" 5. } }
{ 1. 0. } { }
DUP INFORM DROP
→ HT
« ERASE HT 1.
GET SORT DUP
SIZE GET -0.5
SWAP XRNG HT 2.
GET SORT DUP SIZE
GET -0.5 SWAP YRNG
{ # 0d # 0d }
PVIEW HT
OBJ→ DROP
2.
« R→C
» DOLIST
1.
« PIXON
» DOSUBS
DRAX
» 0. WAIT DROP
»

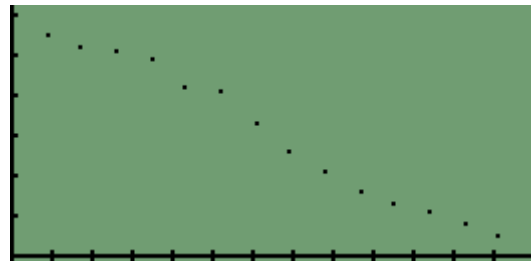
```

Ejecutando el código:


```

DATOS
h= ( 0. 300. 600. 900.
T= ( 38. 34. 32. 31. 31
Alturas en msnm en { }
EDIT          CANCL OK

```



COMENTARIOS:

Es un programa muy sencillo pero es la base para construir programas muy poderosos cuando se trate de manejar datos estadísticos o variables x e y en general.

A esta altura del manual solo haré comentarios sobre los **comandos nuevos** dentro del código:

1. **XRNG** lo que hace es fijar el rango de visualización del eje horizontal en este caso alturas en metros sobre el nivel del mar utilizando los siguientes argumentos:
 - un objeto complejo de la forma (x,y) donde x es el menor valor que tomara el eje x e y el valor mayor que tomara el eje x.
2. **YRNG** lo que hace es fijar el rango de visualización del eje Y en este caso temperaturas en grados centígrados utilizando los siguientes argumentos:
 - un objeto complejo de la forma (x,y) donde x es el menor valor que tomara el eje y e y el valor mayor que tomara el eje y.
3. **R→C** Toma 2 valores reales de la pila y los transforma en complejo (x,y).

```

RAD XYZ DEC R~ 'X'
{HOME TEMP}
5:
4:
3:
2: 1.
1: 2.
INI

```

```

RAD XYZ DEC R~ 'X'
{HOME TEMP}
5:
4:
3:
2:
1: (1.,2.)
RE IM C+R R+C ABS ARG

```

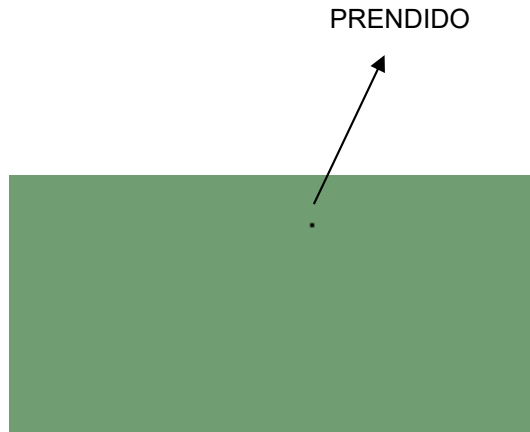
4. **PIXON** Toma un complejo como el que aparece en la pantalla anterior y “prende” el píxel correspondiente en el entorno **PICT**

```

RAD XYZ DEC R~ 'X'
{HOME TEMP}
-----
0:
4:
:
:
:
1: (1.,2.)
RE IM C+R R+C ABS ARG

```

Ejecutamos PIXON



PRENDIDO

5. **DRAX** No necesita argumento alguno para ejecutarlo y lo que hace es dibujar los ejes x e y en el entorno **PICT**.

EJEMPLO 3:

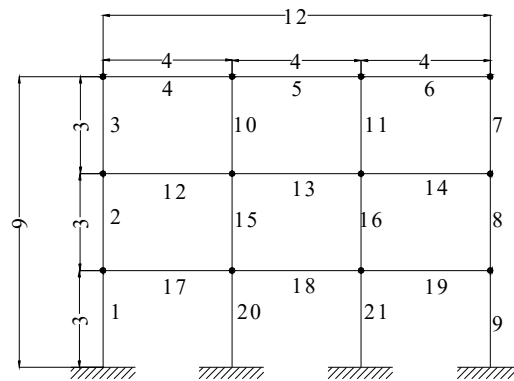
Supóngase que queremos escribir un programa que resuelva estructuras (pórticos) en 2 dimensiones que dibuje la estructura que deseamos resolver. Escribamos el código que dibuja la estructura: El código además pide las dimensiones de cada elemento del pórtico. El Dimensionamiento del entorno **PICT** se hace de manera similar al ejemplo anterior.

```

« ERASE { } 'DATA' STO -.5
"Altura" "" INPUT OBJ→ .5 +
YRNG -.5 "Ancho" "" INPUT OBJ→
.5 + XRNG 1.
"Cuantos elementos son?" "" INPUT OBJ→
FOR i "Datos elemento "
i →STR +
{ { "COORD" "COORD INIC. Y FINAL. EN {}" 5. }
{ "B H L" "BASE ALTURA Y LONGITUD" 5. } }
1. { } DUP INFORM DROP →STR DATA +
'DATA' STO
NEXT
TEXT { # 0d # 0d } PVIEW
DATA SIZE 1. SWAP
FOR i DATA REVLIST i
GET OBJ→ OBJ→ DROP
DROP → E1
« E1 1. GET E1 2.
GET R→C E1 3. GET
E1 4. GET R→C LINE
.2 WAIT
»
NEXT -40. CF { PPAR }
PURGE 0. WAIT
»

```

Veamos el pórtico que queremos resolver:



TODAS LAS DISTANCIAS EN METROS
TODAS LAS SECCIONES DE 0.30 X 0.4

Los elementos ya están numerados como se puede apreciar:
Ejecutemos el código con origen de coordenadas en el apoyo 1

```
RAD XYZ DEC R~ 'X' PRG
{HOME TEMP}
Altura
94
DATA INI
```

```
RAD XYZ DEC R~ 'X' PRG
{HOME TEMP}
Ancho
12
PPAR DATA INI
```

```
RAD XYZ DEC R~ 'X' PRG
{HOME TEMP}
Cuantos elementos son?
214
PPAR DATA INI
```

```
Datos elemento 1.
COOR ( 0. 0. 0. 3. )
B H L ( .3 .4 3. )
COORD INIC. Y FINAL. EN { }
EDIT CANCL OK
```

```
Datos elemento 2.
COOR ( 0. 3. 0. 6. )
B H L ( .3 .4 3. )
COORD INIC. Y FINAL. EN { }
EDIT CANCL OK
```

```

Datos elemento 3.
COOR ( 0. 6. 0. 9. )
B H L ( .3 .4 3. )

BASE ALTURA Y LONGITUD
EDIT | | | | CANCL | OK

```

```

Datos elemento 4.
COOR ( 0. 9. 4. 9. )
B H L ( .3 .4 4. )

COORD INIC. Y FINAL. EN {}
EDIT | | | | CANCL | OK

```

```

Datos elemento 5.
COOR ( 4. 9. 8. 9. )
B H L ( .3 .4 4. )

COORD INIC. Y FINAL. EN {}
EDIT | | | | CANCL | OK

```

```

Datos elemento 6.
COOR ( 8. 9. 12. 9. )
B H L ( .3 .4 4. )

COORD INIC. Y FINAL. EN {}
EDIT | | | | CANCL | OK

```

```

Datos elemento 7.
COOR ( 12. 9. 12. 6. )
B H L ( .3 .4 3. )

COORD INIC. Y FINAL. EN {}
EDIT | | | | CANCL | OK

```

```

Datos elemento 8.
COOR ( 12. 6. 12. 3. )
B H L ( .3 .4 3. )

BASE ALTURA Y LONGITUD
EDIT | | | | CANCL | OK

```

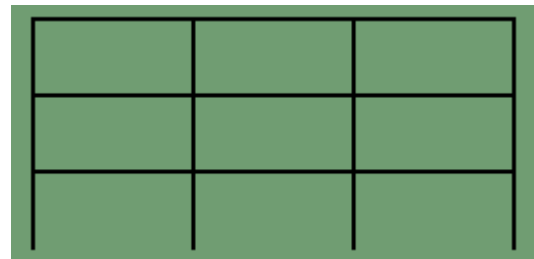
Así sucesivamente hasta llegar el elemento No 21 y entonces veremos:

```

Datos elemento 21.
COOR ( 8. 3. 8. 0. )
B H L ( .3 .4 3. )

COORD INIC. Y FINAL. EN {}
EDIT | | | | CANCL | OK

```



COMENTARIOS:
Ninguno.

15.5 COMANDOS QUE PERMITEN OPERAR SOBRE EL ENTORNO PICT

De estos comandos ya hemos citado algunos. Veamos otros:

15.5.1 BLANK:

Crea un GROB en blanco en la pila del tamaño #nx puesto en el nivel 2 por #ny puesto en el nivel 1. Ej: Creemos un grafico de 20 de ancho por 30 de alto

```

RAD XYZ DEC R~ 'X'
[HOME TEMP]
5:
4:
3:
2: # 20d
1: # 30d
~GROB|BLANK| GOR | GXOR | SUB | REPL

```

```

RAD XYZ DEC R~ 'X'
[HOME TEMP]
5:
4:
3:
2:
1: Graphic 20 x 30
~GROB|BLANK| GOR | GXOR | SUB | REPL

```

Ejecutamos BLANK.

15.5.2 →LCD

Toma un GROB de la pila y lo muestra en la pantalla sobrescribiendo esta totalmente. Ej: Suponiendo que el GROB es un círculo:

```

RAD XYZ DEC R~ 'X'
[HOME TEMP]
5:
4:
3:
2:
1: Graphic 27 x 26
~LCD | LCD+ | SIZE | ANIMA | PRG

```



Ejecutamos →LCD

15.5.3 LCD→

Devuelve a la pila el GROB de la pantalla actual: No requiere poner ningún argumento en la pila:

```

RAD XYZ DEC R~ 'X'
[HOME TEMP]
5:
4:
3:
2:
1: Graphic 131 x 64
~LCD | LCD+ | SIZE | ANIMA | PRG

```

NOTA: Ejecuten ahora →LCD ¿Qué pasa?

15.5.4 ARC:

Toma 3 argumentos de la pila:

- NIVEL 4: El centro del arco de la forma (x,y)
- NIVEL 3: Radio del arco, un real
- NIVEL 2: Un angulo inicial, un real
- NIVEL 1: Un angulo final, un real.

Ejemplo:

Veamos el siguiente codigo:

```
« ERASE { #0 #0 } PVIEW  
(0,0)  
3  
0  
45  
ARC  
0 WAIT  
»
```

Ejecutemos este código:



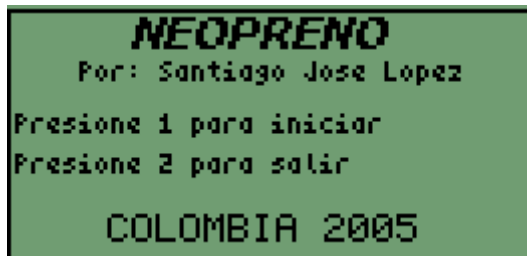
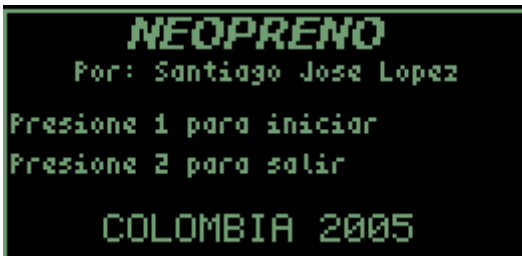
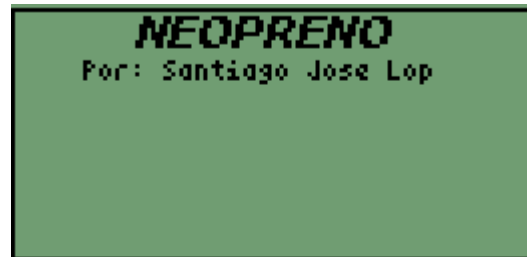
Cambiamos el Angulo a de 45 a 270:



Para finalizar este capitulo escribamos la presentación para un programa que diseña el Neopreno para los apoyos de las vigas en el estribo de un puente y donde se utilizan algunos comandos del entorno PICT (por ejemplo):

```
« ERASE  
{ #0 #0 } PVIEW  
(-6.5,3.1) (6.5,-3.1)  
BOX  
(-3.5,-2) 'CO' STO
```

```
1 26 START
PICT CO DUP (0,0.2) +
'CO' STO
GRAF REPL
NEXT
PICT
(-3.6,2.1)
#70 #50 BLANK
REPL
(-4.7,1.8) 'CO' STO
"Por: Santiago Jose Lopez"
→ n
« 1 24 FOR i
PICT CO DUP
(0.4,0) +
'CO'
STO
n
i DUP
SUB
→STR 1
→GROB REPL
NEXT
»
PICT
(-6.3,0.5)
"Presione 1 para iniciar"
1 →GROB
REPL
PICT
(-6.3,-0.5)
"Presione 2 para salir"
1 →GROB
REPL
PICT
(-4,-2)
"COLOMBIA 2005"
2 →GROB
REPL
{ CO PPAR } PURGE
DO PICT NEG
0.2 WAIT
UNTIL KEY END
»
Ejecutando el código
```



Se ven estas dos ultimas pantallas hasta que se pulsa una tecla.

COMENTARIOS NINGUNO.

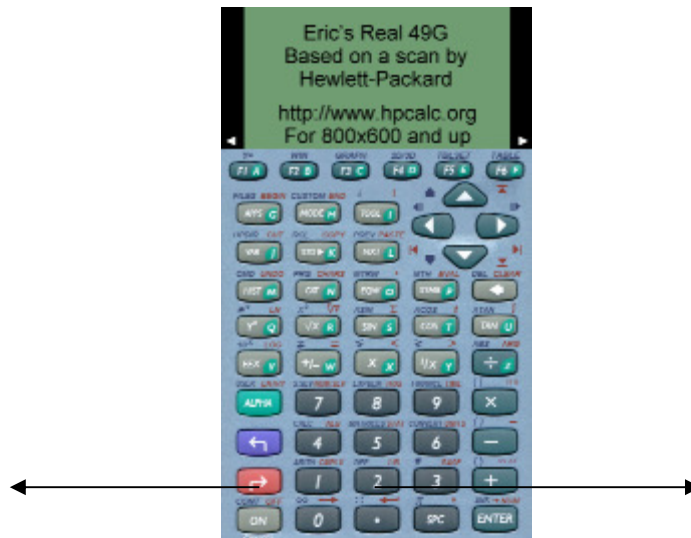

```
DEG XYZ DEC R= 'X'  
CHOME DEOPREDS  
  
LIBRERIAS  
  
A | D | G | A | R | 18
```

16. LIBRERÍAS.

16. LIBRERÍAS.

16.1 DEFINICIÓN

Una librería es un objeto que contiene objetos que tienen asignado un nombre y que pueden funcionar como extensiones del conjunto de comandos incorporados, es decir que funciona como un comando más de la calculadora. Una librería puede residir en cualquiera de los puertos disponibles en la HP a menos que esta especifique lo contrario. Los puertos disponibles en la HP49G son el 0, 1, 2 y los podemos visualizar pulsando la tecla de cambio derecha y luego la tecla donde está el 2:



Presionando estas dos teclas veremos lo siguiente:



Donde podemos apreciar los puertos 0, 1, 2 y también vemos que hay tres librerías instaladas **GMENU**, **Jazz**, **AASHTO**.

Si queremos saber en qué puerto está instalada cualquiera de estas librerías buscamos en cada uno de los puertos. Para esto pulsamos la tecla de menú correspondiente a cada uno de los puertos hasta encontrar la librería que buscamos.

16.2 COMO INSTALAR Y DESINSTALAR LIBRERÍAS.

Para explicar esto veamos paso a paso la que hay que hacer para **INSTALAR** una librería:

1. Ponemos la librería que queremos instalar en el nivel 1 de la pila:

```
DEG XYZ DEC R~ 'X'
CHOME3
5:
4:
3:
2:
1: Library 1570: AASH...
379 393 392
```

2. Ponemos en la pila el numero del puerto donde queremos instalar dicha librería, en este caso elegimos el 0:

```
DEG XYZ DEC R~ 'X'
CHOME3
5:
4:
3:
2: Library 1570: AASH...
1: 0.
379 393 392
```

3. Pulsamos la tecla **STO** (segunda tecla tercera fila) y desaparecerá la librería de la pila:

```
DEG XYZ DEC R~ 'X'
CHOME3
5:
4:
3:
2:
1:
1570 379 393 392
```

4. Ahora manteniendo presionada la tecla **ON** oprimimos la tecla **C** (tercera tecla primera fila) para así reiniciar la HP y pegar la librería en la RAM de la HP.

De esta manera ya tendremos nuestra librería instalada. Para ver si el procedimiento fue exitoso pulsamos la tecla de cambio derecha y la tecla que contiene el numero 2 y veremos que nuestra librería ya esta instalada:

```
DEG XYZ DEC R~ 'X'
[HOME]
5:
4:
3:
2:
1:
[GMENU] Jazz [AASHT] :0: :1: :2:
```

Ahora vemos como **DESINSTALAR** librerías suponiendo que vamos a desinstalar la librería que acabamos de instalar:

1. Averiguamos cual es el numero de la librería que vamos a desinstalar entrando al puerto donde la instalamos presionando la tecla de cambio derecha, la tecla que contiene el numero 2 y la tecla de menú correspondiente al puerto en este caso 0:

```
DEG XYZ DEC R~ 'X'
[HOME]
5:
4:
3:
2:
1:
[GMENU] Jazz [AASHT] :0: :1: :2:
```

```
DEG XYZ DEC R~ 'X'
[HOME]
5:
4:
3:
2:
1:
1570 979 993 992
```

Presionamos la tecla de menú 1570 para saber si esta es la librería a desinstalar:

```
DEG XYZ DEC R~ 'X'
[HOME]
5:
4:
3:
2:
1: Library 1570: AASH...
1570 979 993 992
```

En efecto vemos que se trata de la la librería AASHTO.

2. Ya teniendo el numero (1570) borramos de la pila la librería y ponemos el numero del puerto y el numero de la librería así:

```
DEG XYZ DEC R~ 'X'
[HOME]
5:
4:
3:
2:
1:                                0: 1570.
1570 979 993 992
```

Lo duplicamos presionando ENTER (equivalente en este caso a un DUP)

```
DEG XYZ DEC R~ 'X'
[HOME]
5:
4:
3:
2:                                0: 1570.
1:                                0: 1570.
1570 979 993 992
```

Ejecutamos el comando **DETACH** (para despegar la librería de la RAM de la HP) escribiéndolo en la pila o buscándolo en el CAT. En este caso lo escribiremos en la pila:

```
DEG XYZ DEC R~ 'X'
[HOME]
4:
3:
2:
1:                                0: 1570.
DETACH
1570 979 993 992
```

```
DEG XYZ DEC R~ 'X'
[HOME]
5:
4:
3:
2:                                0: 1570.
1:                                0: 1570.
1570 979 993 992
```

PULSAMOS ENTER

3. Ejecutamos el comando **PURGE** escribiéndolo en la pila y listo, ya esta nuestra librería desinstalada:

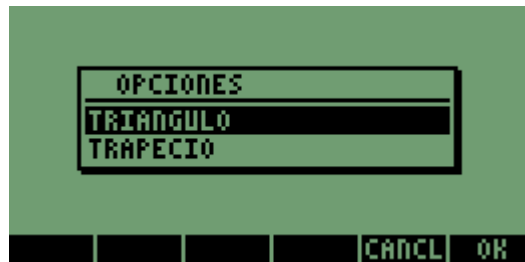
```
DEG XYZ DEC R~ 'X'
[HOME]
5:
4:
3:
2:
1:
GMENU Jazz :0: :1: :2:
```

16.3 COMO CREAR LIBRERÍAS

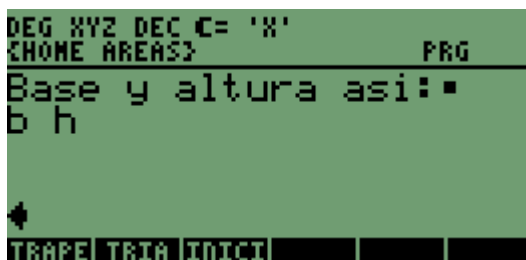
Para crear librerías existen muchos programas como LIB, LIBKIT entre otros. Por supuesto que nosotros también podemos escribir un programa que cree librerías pero tendríamos que tener conocimientos de programación en lenguajes de mas bajo nivel lo que no concierne a este libro. Para el caso utilizaremos un programa llamado LIBKIT V1.1 escrito por el señor Dante Camargo el cual sirve para crear librerías, instalar y desinstalar librerías entre otras cosas. El programa que pasaremos a librería será un pequeño programa que contiene 3 variables el cual sirve para calcular las áreas de dos figuras geométricas; un triangulo y un trapecio. Las variables son como sigue (primero esta el nombre de la variable y luego la tarea que ejecuta):

- INICIO: es la entrada al programa y contiene la presentación de este y las opciones
- TRIA: Calcula el área del triangulo en función de su altura.
- TRAPE: calcula el área del trapecio en función de sus bases.

NOTA: Esto es solo un ejemplo ya que es un programa supremamente sencillo y además se puede escribir en una sola variable si se quiere así. Veamos el programa:



Esto en el caso de que se eligiera triangulo.



```

DEG XYZ DEC C= 'X'
[HOME AREAS]
-----
5:
4:
3:
2:
1: Area= : 25.
TRAPE TRIA INICI

```

Para Trapecio:

```

DEG XYZ DEC C= 'X'
[HOME AREAS] PRG
-----
Base mayor, base menor...
y altura así:
B b h
10 8 3
TRAPE TRIA INICI

```

```

DEG XYZ DEC C= 'X'
[HOME AREAS] PRG
-----
QUE NOMBRE TENDRA SU LIBRERIA?
PUDE SER CUALQUIERA, MIENTRAS
NO EXISTAN ESPACIOS.
4
2 [XXXXX] INICI TRIA TRAPE
TRAPE TRIA INICI

```

Ahora veamos como se crea la librería:

1. Debemos tener un directorio donde se encuentre el programa con todas sus subrutinas:

```

DEG XYZ DEC C= 'X'
[HOME AREAS]
-----
5:
4:
3:
2:
1:
INICI TRIA TRAPE

```

Aquí estamos en el directorio ÁREAS (Véase la parte superior de la pantalla.)

2. Ahora corremos el programa libkit:

```

DEG [LIBKIT V1.1]
[CHOM]
-----
5: CREAMA UNA LIBRERIA
4: INSTALAR UNA LIB
3: DESINSTALAR UNA LIB
2: CONVERTIR LIB+DIR
1: RENUMERAR UNA LIB
1: SALIR
[CANCL] [OK]

```

PRESIONAMOS OK

```

PARA CREAR UNA LIBRERIA . . .
NECESITA TENER UN DIRECTORIO
CON LOS ARCHIVOS O VARIABLES
QUE CONTENDRA SU LIBRERIA.
LUEGO DEBERA SITUARSE DENTRO
DE ESE DIRECTORIO.
[GRAPH] [OK]

```

PRESIONAMOS OK

```

DEG XYZ DEC C= 'X'
[HOME AREAS] PRG
-----
QUE NOMBRE TENDRA SU LIBRERIA?
PUDE SER CUALQUIERA, MIENTRAS
NO EXISTAN ESPACIOS.
AREAS1
2 |XXXXX|INICI|TRIA|TRAPE|

```

EL NOMBRE ES AREAS1 ENTER

```

DEG XYZ DEC C= 'X'
[HOME AREAS] PRG
-----
QUE NUMERO TENDRA SU LIBRERIA?
PUEDE SER ENTRE: 769 a 1792.
1500
$TITL 2 |XXXXX|INICI|TRIA|TRAPE|

```

1500 POR EJEMPLO. ENTER

```

DEG XYZ DEC C= 'X'
[HOME AREAS] PRG
-----
SELECCIONE LAS VARIABLES QUE
SI APARECERAN EN EL MENU?
< INICIO >
$CONF $RONI $TITL 2 |XXXXX|INICI|

```

PRESIONAMOS ENTER

```

DEG XYZ DEC C= 'X'
[HOME AREAS] PRG
-----
SELECCIONE LAS VARIABLES QUE
NO APARECERAN EN EL MENU?
SI NO HAY PRESIONE [ENTER]...
< TRIA TRAPE >
INICI|TRIA|TRAPE|

```

TRIA TRAPE PRESIONAMOS ENTER

```

DEG XYZ DEC C= 'X'
[HOME AREAS] PRG
-----
SU LIBRERIA HA SIDO CREADA
CON EXITO!!!
DESEA INSTALARLA AHORA O
PREFIERE GUARDARLA COMO UNA
VARIABLE???:
INSTALAR
INSTA|GUAR|

```

PRESIONAMOS INSTA ENTER

```

DEG XYZ DEC C= 'X'
[HOME AREAS] PRG
-----
INSTALAR EN...?
SU
CO PUERTO 0
DE PUERTO 1
PR PUERTO 2
VARIABLE:::
INSTALAR 4
|CANCL|OK|

```

ELEGIMOS PUERTO 0 Y OK

Se apagara la HP. La prendemos y ya tenemos nuestra librería creada:

```

DEG XYZ DEC C= 'X'
[HOME AREAS]
7:
6:
5:
4:
3:
2:
1:
AREAS|GMENU|Jazz|LIBRI|:0:|:1:|

```


COMENTARIOS FINALES

Bueno, hemos llegado al final de este pequeño libro pero espero que les haya sido de utilidad tanto a estudiantes como a profesionales de las diferentes ramas de la ingeniería.

Como comentario final quiero agregar que para nosotros los Ingenieros siempre es importante y siempre va a ser importante tener conocimientos de programación así sean mínimos ya que en este momento de mi vida me doy cuenta de que la programación nos enseña muchísimas cosas, nos hace mas inteligentes y analíticos, nos enseña a pensar de una manera estructurada y lógica, nos enseña a crear algoritmos en nuestras mentes no solo para solucionar problemas ingenieriles sino también de nuestra vida cotidiana.

Definitivamente la programación es una herramienta tan poderosa que simplemente nos divide entre quienes saben y los que no saben programar y con esto no me refiero solo a las Hp sino también a aplicaciones creadas en otros lenguajes de programación en los cuales me estoy adentrando en este momento y me han dejado ver que la programación para mi ha dejado de ser un mundo para convertirse en un universo.

Cualquier inquietud no duden en comunicarse conmigo a andresgarcia35@yahoo.es

ING. ANDRÉS R GARCÍA M

BOGOTA JULIO 7 DE 2008

BIBLIOGRAFÍA:

- Guía del usuario de la Calculadora HP 48G y 49G.

BIBLIOGRAFIA WEB:

- www.hpcalc.org

- www.adictoshp.org