Programación

Sintaxis:

- « declaración de variables locales « programa » »
  - Símbolos « » : ↱ +
  - Declaración de variables locales: valor_inicial_A valor_inicial_B valor_inicial_C ➔ A B C
    - ✱ Las variables locales no se guardan en el directorio de trabajo que tengamos activo. Actúan por encima de las variables globales que se llamen igual, no las sobrescriben.
    - ✱ Las variables globales (no locales) que almacenemos dentro del programa quedarán guardadas en el directorio de trabajo activo a menos que las borremos antes de cerrar el programa.
    - ✱ Los valores iniciales se pueden omitir y pasarlos desde la pila antes de llamar al programa.
    - ✱ Ej: « ➔ A B C « A B C ∗∗ »
- Lo editamos y guardamos igual que el resto de variables.
- Desde un programa podemos llamar a otros programas que hayamos creado siempre que sean visibles desde el directorio de trabajo que tengamos activo. Se pueden entender como subrutinas.
- Si escribimos **OFF** en mitad de un programa la calculadora se apagará, y continuará ejecutando el programa cuando la encendamos de nuevo.

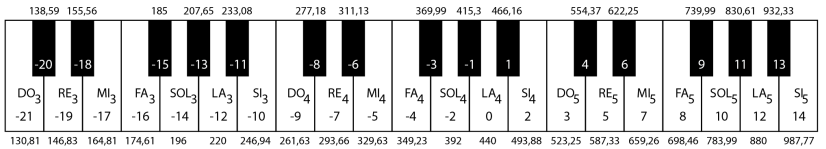
Introducción de datos:

- “Texto informativo:” **PROMPT**
- “Texto informativo:” “Valor predeterminado” **INPUT** (Para captura de números **OBJ➔**)
- El método **INFORM**:
  - **“Título”**
  - **{ { “Var1” “Texto1” tipo_obj1 } { “Var2” “Texto2” tipo_obj2 } { “Var3” “Texto3” tipo_obj3 } { “Var4” “Texto4” tipo_obj4 } }**
  - **{ num_cols separación_texto_valor}**: opciones de formato
  - **{ Var1reset Var2reset Var3reset Var4reset }**: Valores de reconfiguración al pulsar RESET
  - **{ Var1ini Var2ini Var3ini Var4ini }**: Valores iniciales que aparecen al mostrar el INFORM
  - **INFORM**
  - Tras pulsar OK, devuelve una lista con los valores definidos ordenados según los hayamos definido, y un número 1 de verificación.
  - Si pulsamos CANCEL, no devuelve la lista y sí un 0 de verificación.
- El método **CHOOSE**:
  - **“Título”**
  - **{ { “Texto1” Valor1 } { “Texto2” Valor2 } { “Texto3” Valor3 } }**

  - **num**: Número del elemento inicialmente marcado.
  - **CHOOSE**
  - Devuelve 0 si se cancela. Si pulsamos OK devuelve 1 en el primer nivel de la pila y el valor de la selección en el segundo.
  - Se puede utilizar una lista como { Elem1 Elem2 Elem3 }, y tanto el texto a mostrar como el valor a utilizar serán iguales y de valor Elem1.

Avisos:

- “Texto a mostrar” **MSGBOX**
- frecuencia tiempo **BEEP**: Pitido. La frecuencia se da en Hz y el tiempo en segundos.
  - Las notas musicales: se define LA como f_{LA}=440Hz. La frecuencia de cada semitono se calcula como $f_I = f_{LA} \cdot 2^{I/12}$, donde I es el salto entre LA_4 y el semitono a calcular.



- tiempo **WAIT**: detiene la ejecución de un programa durante el tiempo que se especifique (en segundos). Si en tiempo se pone -1, esperará hasta que el usuario presiones una tecla.

Errores:

- Si creemos que en una parte del programa que estemos creando se puede dar un error, podemos considerarlo como un código trampa que si falla, en vez de mostrar el aviso por defecto del sistema, hará lo que nosotros programemos.
  - **IFERR** código_trampa **THEN** código_error **ELSE** código_normal **END**
  - “Aviso de error” o número o 0 **DOERR**
    - ✱ Con una cadena de caracteres mostraremos nuestro propio mensaje de error.
    - ✱ Con un número real se mostrará el error definido por defecto para ese número.
    - ✱ Con 0 provoca un error sin actualizar el número de error.
    - ✱ Si DOERR está fuera de un código trampa, detiene la ejecución del programa.
  - **ERRN**: devuelve el número de error del error más reciente como un entero binario.
  - **ERRM**: Muestra el mensaje de error del error más reciente.
  - **ERR0** (cero): Borra el número de error del error más reciente y guarda un 0 en su lugar.

Modificación de banderas (Flags):

- ¡Las que vemos en el menú de banderas son negativas!
- num **CF**: Desactiva (Clear) la bandera seleccionada.
- num **SF**: Activa (Set) la bandera seleccionada.

- **RCLF** (ReCall Flags): Lista de 4 enteros binarios de 64-bit con el estado actual de las banderas.
- {lista_anterior} **STOF** (STOre Flags): devuelve al estado listado las banderas.
- Condiciones lógicas de banderas: **FS?**, **FC?**, **FS?C**, **FC?C**

Condicionales:

- condición **IF THEN** código1 **ELSE** código2 **END**
- **IF** 'condición' **THEN** código1 **ELSE** código2 **END**
- condición código_verdadero código_falso **IFTE**
- **CASE** condición1 **THEN** código1 **END** condición2 **THEN** código2 **END** código_alternativo **END**
- **CASE** 'condición1' **THEN** código1 **END** 'condición2' **THEN** código2 **END** código_alternativo **END**
  - Se pueden poner tantas condiciones como se quieran.
  - Se ejecuta solo la primera que se cumpla.
- Tipos de condiciones lógicas: **==, ≠, <, >, ≤, ≥, AND, OR, XOR, NOT, SAME**
- 'expr_alg' 'variabe' **LININ**: comprueba si la expresión algebraica es lineal en la variable dada.

Bucles:

- inicio fin **FOR** índice código **NEXT** o … incremento **STEP**
- inicio fin **START** código **NEXT** o … incremento **STEP**
- **DO** código condición **UNTIL END**
- **DO** código **UNTIL** 'condición' **END**
- **WHILE** condición **REPEAT** código **END**
- **WHILE** 'condición' **REPEAT** código **END**
- Uso de **INCR** y **DECR**:
  - 'variable' **INCR**: variable 1 + 'variable' STO variable
  - 'variable' **DECR**: variable 1 - 'variable' STO variable
- El comando **KEY** se puede introducir en bucles para salir de los mismos a petición del usuario. KEY devuelve 0 la primera vez que se escribe. Cada vez que desde entonces se llame al comando KEY, este devolverá un 0 si ninguna tecla se ha pulsado desde el paso anterior y un 1 en caso contrario. Sirve para salir de los bucles sin tener que detener la ejecución del programa por completo.

Resultados:

- Los resultados se pueden mostrar tal cual en la pila en un orden determinado de tal manera que quien ha hecho el programa entienda lo que está viendo. En cambio, cuando hacemos programas que vayamos a compartir es aconsejable dar información sobre los resultados a los que llegamos.
- Podemos:
  - Utilizar la estructura **INFORM**.
  - Etiquetar los resultados: num "etiqueta" **→TAG**

  - Guardarlos en variables globales.
  - Mostrarlos con **MSGBOX**.

Testear programas:

- Para ver paso a paso lo que hace un programa:
  - **Poner programa en el primer nivel de la pila.**
  - Menú de programación: ↰ **EVAL**
  - **15. RUN & DEBUG**
  - **DBUG**
  - **SST↓**
- Podemos detener la ejecución de un programa con **HALT** en mitad del mismo para continuar paso a paso a partir de ahí. Cuando queramos continuar pulsamos **CONT**: ↰ **ON**
- Si lo que queremos es matar el proceso definitivamente: **KILL**

Menús:

- Podemos crear menús usando la sintaxis mostrada a continuación:
- { {"nombre1" «programa1»} {"nombre2" «programa2»} } **MENU**

Tipos de objetos:

- El comando **TYPE** devuelve el tipo de objeto en el primer nivel de la pila

Tipo	Num.	Tipo	Num.	Tipo	Num.
U: Número real	0	U: Objeto gráfico	11	S: Complejo extendido	22
U: Número complejo	1	U: Objeto etiquetado (tagged)	12	S: Matriz enlazada	23
U: Cadena de caracteres	2	U: Objeto con unidades	13	S: Caracter	24
U: Matriz (o vector) real	3	U: Nombre XLIB	14	S: Objeto de código	25
U: Matriz (o vector) compleja	4	U: Objeto de directorio	15	S: Dato de librería	26
U: Lista	5	U: Librería	16	S: Fuente (tipografía) pequeña	27
U: Variable global	6	U: Objeto de backup	17	U: Entero real	28
U: Variable local	7	B: Función tipo built-in	18	U: Matriz (vector) simbólica	29
U: Programa	8	B: Comando tipo built-in	19	S: Fuente (tipografía)	30
U: Objeto Algebraico	9	S: Binario de sistema	20	S: Objeto extendido	31
U: Entero binario	10	S: Real Extendido	21		

- U: Objetos de usuario. S: Objetos del sistema. B: Comandos built-in.

Autor:

- Ion Elberdin Navarro.
- ionelberdin@gmail.com
- http://ionelberdin.com (actualizaciones de este manual y otros)