Capítulo 10

Introducción a la programación

Tabla de contenidos

Para mayor información, diríjase a http://www.hp.com/calculators/hp49

Introducción

Este capítulo describe cómo crear y ejecutar programas con la HP 49G. La HP 49G ofrece un ambiente propicio para la programación. Los programas pueden oscilar en complejidad desde una tarea simple tal como realizar una serie de operaciones aritméticas a un proceso complejo que requiere entrada de datos, procesamiento intensivo y resultados con salida en formato gráfico.

Este capítulo enfoca la creación y ejecución de programas en modo algebraico solamente. Vea la Guía del Usuario Avanzado *(Advanced User's Guide)* por infomación sobre cómo crear y utilizar programas en el modo RPN. Vea la Guía de Bolsillo *(Pocket Guide)* por detalles sobre los comandos de programación disponibles.

Para comenzar

Esta sección contiene un ejemplo sobre cómo crear un programa simple para calcular la hipotenusa de un triángulo rectángulo, utilizando el teorema de Pitágoras. Al iniciar el programa usted suministra las longitudes de los lados conocidos en forma de argumentos.

Este programa es un ejemplo de un simple cálculo algebraico, utilizando los argumentos especificados por usted. Dentro del programa, el cálculo se encierra entre comillas simples (Ⓡ⌐) para delimitarlo como un objeto algebraico. Si necesita realizar un procedimiento intensivo que incluya bucles o bifurcaciones, necesitará un procedimiento anidado (Ⓡ«») para delimitarlo desde los argumentos.

El programa realiza lo siguiente:

- Reune las longitudes de los lados conocidos como agumentos y los almacena como variables locales, es decir aquellas que sólo existen mientras el programa se encuentra en ejecución.
- Utiliza las variables para calcular la longitud de la hipotenusa y devuelve los resultados a la historia.

Cree el programa de la siguiente forma:

1. Ubique los delimitadores en la línea de comando.
   Ⓡ«» «»
2. Defina las dos variables locales para aceptar los argumentos para la longitud del lado.
   ⓁⓇ(ALPHA)A(SPC)(ALPHA)B(SPC) « → A B »

3. Defina la ecuación para calcular la hipotenusa.
   Observe que necesita utilizar ⁽ para encerrar la ecuación y separarla de la definición de los argumentos.
   ⁽√ₓ⁾()ALPHAAy^x2+ALPHABy^x2 « → A B '√(A^2+B^2)' »
4. Mueva el cursor fuera del programa y especifique que desea almacenar el programa como “PYTH”.
   ▼STOALPHAALPHAPYTH
   « →A B '√(A^2+B^2)' » ▶ PYTH
5. Pulse ENTER para almacenar el programa.
   ENTER

Al ejecutar el programa, usted especifica las longitudes de los lados como argumentos del mismo. Por ejemplo, para poner en ejecución el programa para calcular la hipotenusa de un triángulo rectángulo con lados equivalentes a 3 y 4 unidades:

1. Visualice una lista de las variables en el directorio.
   VAR
2. Pulse la tecla de función correspondiente a su programa. El nombre del programa se inserta en la línea de comandos, con paréntesis preparados para los argumentos.
3. Introduzca los argumentos, separados por una ,, entre los paréntesis.
   3▶,4
4. Pulse ENTER para calcular la hipotenusa.
   ENTER
   El resultado se devuelve a la línea de comandos.

Cómo crear, archivar y poner en ejecución un programa

Un programa es un objeto que puede almacenarse en una variable. Es decir: usted crea un programa, le asigna un nombre y lo archiva en un directorio.

- Para crear un programa, pulse ▶«». Aparecen en la línea de comandos los delimitadores de programa listos para que introduzca cifras, al tiempo que el anunciador PRG aparece en la parte superior de la pantalla para indicar que está en modo de programa.
  Para crear el programa, utilice las funciones del teclado y las teclas de operación y seleccione los comandos del menú de programación.

A medida que selecciona las teclas de función y operación, las funciones y operaciones aparecen en el programa.

Utilice **;** para separar funciones y cálculos dentro de un procedimiento anidado. Para introducir **;**, pulse y mantenga pulsada ⓡ, y pulse (.SPC).

Para facilitar la lectura, puede utilizar ⓡ↵ para incorporar cortes de renglón.

Por detalles sobre edición de programas, por ejemplo: cortar, copiar y pegar, vea "Cómo editar la línea de comandos", en el capítulo 2 "Operaciones básicas".

Para incorporar comentarios a su programa:

– Para toda una línea de comentarios, comience la misma con un caracter @ (ⓐ(ALPHA)ⓛ(ENTER)).
– Para un segmento comentado dentro de un código, encierre el mismo entre caracteres @.

• Para archivar el programa:

a. Pulse ⓡ▼ para trasladar el cursor pasado el final del mismo.
b. Pulse (STO▶) para insertar el símbolo ▶ pasado el programa.
c. Introduzca un nombre para el programa y pulse (ENTER).

• Para poner en ejecución un programa, se accede simplemente al directorio donde éste reside y: o introduce el nombre del programa en la línea de comandos o pulsa (VAR) y lo selecciona del menú.

Al seleccionar el programa del menú, la HP 49G lo reconoce como tal e inserta su nombre en la línea de comandos, con paréntesis vacíos listos para que usted introduzca los argumentos. Después de hacerlo, pulse (ENTER) para ejecutar el programa.

El menú de programación

El menú de programación contiene los comandos que puede utilizar en un programa. Seleccione una categoría para mostrar los comandos disponibles a la misma. Del menú se seleccionan comandos para incluirlos en el programa. El menú de programación es sólo una ayuda para escribirlos; necesita conocer la sintaxis de los comandos y cómo utilizarlos en el programa. Vea la guía de bolsillo por detalles sobre los comandos de programacióny su sintaxis.

Examinar el menú de programación servirá para tener una idea de los tipos de operaciones programáticas disponibles a la HP 49G.

• Para visualizar el menú de programación, pulse ⓛ(PRG).

Modos algebraicos y RPN

En el modo RPN, la HP 49G utiliza intensamente la pila. Al programar en modo RPN se utiliza la pila para:

- proveer los datos que utiliza el programa
- construir los comandos utilizados por el programa
- mantener la salida generada por el programa.

En el modo algebraico la pila no se encuentra disponible. Se utilizan otros métodos para construir el programa e incorporarle datos.

Cómo utilizar funciones que requieren argumentos

Al utilizar una función que requiere argumentos:

- En el modo RPN, se ubican los argumentos en la pila antes de invocar la función.
- En modo algebraico, usted suministra los argumentos entre paréntesis, después de invocar la función.

Por ejemplo, puede utilizar el comando INPUT para requerir datos. Los siguientes segmentos codificados muestran cómo utilizar el comando INPUT para reunir datos en modo RPN y algebraico.

- En modo RPN el siguiente segmento codificado requiere entrada de datos, los reúne en forma de cadena y los convierte en un número. Al final del proceso, los datos se encuentran en el nivel 1 de la pila:

```
«"ENTER A NUMBER"
""
INPUT
OBJ→»
```

- En modo algebraico, el siguiente segmento codificado lleva a cabo la misma operación. Al final del proceso, los datos se almacenan en una variable global, NUM1, listos para ser utilizados en el programa.

  Observe que al utilizar una variable global y no local, puede continuar la expresión variable con una función.

```
« INPUT ("ENTER A NUMBER","")▶NUM1;
OBJ → (NUM1) ▶NUM1»
```

Manejo de datos

Esta sección describe brevemente cómo puede suministrar datos a sus programas y cómo puede hacer salir los datos que el mismo produce.

Entrada de datos

Puede utilizar uno de los siguientes métodos para especificar los datos que desea que su programa utilice :

- como argumentos al poner en ejecución el programa
- como variables que crea usted en la memoria antes de poner en ejecución el programa
- pidiendo datos para entrar durante la ejecución del programa
  - Vea: “Como utilizar funciones que requieren argumentos” en la página 10-5, con un ejemplo de utilización de la función INPUT para requerir datos.
  - Al utilizar una función tal como INPUT para reunir datos numéricos mientras se ejecuta el programa, los datos se devuelven en forma de cadena. Necesitará convertirlo en un número utilizando una función como OBJ.

Salida de datos

Los datos que salen en modo algebraico se escriben en la historia. Observe los siguientes puntos con respecto a la salida de datos:

- Al completarse el programa, la historia muestra solamente la última salida en el nivel 1. Para retener salidas creadas durante el procesamiento, puede hacer escribir la salida a una variable global u variables locales a medida que el programa progresa. Este método le da la flexibilidad de dar formato a la salida de datos y de agregar comentarios para mejorar la claridad.
- Algunas funciones resultan en valores múltiples. Para esas funciones, los valores se escriben a una lista y, a menos que los haga salir a una variable, la lista aparecerá en la historia.

Cómo fluye un programa

Los programas de la HP 49G tienen un punto de entrada (el principio del programa) y uno de salida (el final del programa). No existe un comando como el GOTO que puede utilizar para saltar a un punto dentro del programa sino que se utilizan estructuras bucle y bifurcaciones como la `IF THEN` para controlar el orden de las operaciones. Vea detalles en “Estructuras condicionales y bucles” en la página 10-12.

Desde dentro de los programas usted puede ir ejecutando otros, de esta forma, puede crear programas modulares. Por ejemplo: podría crear tres programas discretos de componentes llamados: INPUT, PROCESSING, y OUTPUT. Podría entonces crear un programa maestro que ejecute cada uno de estos tres componentes por vez, como sigue:

```
«INPUT PROCESSING OUTPUT»
```

Procedimientos anidados

Si utiliza variables locales para reunir argumentos de entrada, necesita utilizar procedimientos anidados si desea realizar bifurcación y bucle. No puede llevar a cabo bifurcación ni bucle desde dentro de un objeto algebraico.

Para introducir un nuevo procedimiento anidado en su código pulse (↱ «») para introducir los delimitadores. Ingrese el código de procedimiento entre ellos.

Por ejemplo, en el siguiente segmento programático, los argumentos ingresados se asignan a variables A y B. El objeto algebraico, un cálculo que suma las variables, necesita estar encerrado entre comillas simples ya que continúa inmediatamente a la definición de variable local. Este ejemplo devuelve la suma de A y B a la historia.

```
« → A B 'A+B' »
```

En el siguiente segmento programático usted utiliza un procedimiento anidado ya que el procedimiento involucra más que un cálculo simple. Este ejemplo compara A y B, y realiza cálculos basados en la comparación. Los resultados de los cálculos se almacenan en variables globales C y D.

```
« → A B
  « IF A>B
    THEN A-B ▶ C; A^2-B^2 ▶ D
    ELSE B-A ▶ C
    END
  »
»
```

Observe que, dentro de un procedimients anidado, necesita utilizar ; para separar los cálculos. Para insertar el carácter; pulse y mantenga pulsada ↱ y pulse SPC.

Cómo trabajar con variables

Se utilizan las variables para mantener datos dentro de los programas. Existen dos tipos de variables dentro del medio programático de la HP 49G.

- Usted crea **variables locales** dentro de su programa; por ejemplo, las variables locales contienen los valores establecidos por los argumentos que utiliza cuando invoca el programa.

  Un programa sólo puede acceder una variable local dentro del procedimiento anidado en el que fue creado y cualquiera que contenga.

- En un programa, puede usted crear **variables globales** o utilizar las existentes. Vea el capítulo 7, "Cómo almacenar objetos" por detalles sobre cómo crear variables globales. Sírvase observar lo siguiente:
  - Las variables globales están disponibles en cualquier lugar dentro de un programa.
  - Para eliminar una variable global utilizando un código, utilice el comando *PURGE* (Borrar).
  - Si utiliza variables globales en su programa, deben estar ubicadas en el mismo directorio o más arriba, que el programa.

Cómo utilizar variables locales

Hay ciertas limitaciones con las variables locales que debe tener en cuenta, éstas son:

- Inmediatamente después de una expresión variable local, el código programático debe contener ya sea:
  - un cálculo algebraico encerrado entre comillas simples, o
  - un procedimiento anidado, encerrado por « ».
- Se encuentra disponible una variable local en el procedimiento anidado donde fue creada, y en todos los procedimientos anidados que contiene.
- Usted puede crear una variable local con el mismo nombre que una variable global existente (o sea, una variable en el mismo directorio que el programa). Los comandos con el nombre de la variable utilizarán el valor de la variable local en lugar del global.

Cómo configurar variables

Por lo general, usted asigna variables a las entradas o a los resultados de los procesos y cálculos que realiza su programa. Puede utilizar variables

locales para almacenar resultados intermedios que quiere volver a utilizar en subsecuentes procedimientos anidados dentro del programa. Utilice las variables globales para almacenar datos para un acceso más amplio.

Cómo configurar variables locales para contener argumentos de entrada

1. En la línea de comando, ubique el cursor inmediatamente a la derecha de símbolo « de apertura.
2. Pulse ▶ para introducir el símbolo →.
3. Introduzca el nombre de una variable local para cada argumento de entrada utilizado por su programa, separándolos con una SPC.

Por ejemplo, si su programa utiliza dos argumentos y desea asignar el valor de estos argumentos a variables locales A y B, el principio de su programa aparecerá como sigue:

▶ « » ▶ ▶ ALPHA A SPC ALPHA B « → A B

Cómo asignar una variable local a un valor

Después del valor, pulse ▶ para introducir el símbolo →, e ingresar el nombre de la variable local.

Por ejemplo, para hacer que la variable local G contenga 9.81, la aceleración de la gravedad, la variable se crea como sigue:

▶ « » 9.81 ▶ ▶ ALPHA G « 9.81 → G »

El siguiente ejemplo:

- acepta un argumento de entrada
- crea la variable local G
- la multiplica por el argumento y ubica el resultado en la historia.

```
« → A
 « 9.81 → G
  « A*G »
 »
»
```

En el ejemplo siguiente, el cálculo A*G no reconoce la variable local G como 9.81, ya que se encuentra fuera del procedimiento anidado donde se expresó la variable. El cálculo A+G reconoce a G como 9.81

```
« → A
 «9.81 → G 'A+G'»
A*G
»
```

Cómo asignar una variable local al resultado de un cálculo

El siguiente segmento programático muestra cómo asignar una variable local al resultado de un cálculo, y utilizar el resultado en un cálculo subsiguiente. El programa acepta dos argumentos de entrada y los utiliza en los cálculos.

1. En la línea de comandos inserte los delimitadores de programa y especifique las variables locales que contendrán los argumentos de entrada.
   ▶ « » ▶ → ALPHA A SPC ALPHA B SPC `« → A B »`
2. Comience otro procedimiento anidado y defina el cálculo inicial.
   ▶ « » ALPHA A + ALPHA B
   ```
   « → A B
     « A+B »»
   ```
3. Almacene los resultados del cálculo en la variable local C.
   ▶ → ALPHA C
   ```
   « → A B
     « A+B → C »»
   ```
4. Abra un nuevo procedimiento anidado e introduzca un cálculo que utilice los resultados del cálculo inicial.
5. ▶ « » ALPHA C + √x ↶ ( ) ALPHA A − ALPHA B
   ```
   « → A B
     « A+B → C
       « C+√(A-B) »
     »
   »
   ```

Cómo utilizar variables globales

Usted puede utilizar las variables globales existentes en sus programas; éstas son diferentes a las locales por lo siguiente:

- Las variables globales están disponibles a todo el programa, independientemente de los procedimientos anidados.
- En un procedimiento anidado, a diferencia con las variables locales, usted puede crear más de una variable global.

Dentro de un programa, se utiliza la tecla STO▶ para definir una variable global. La tecla STO▶ produce un símbolo ▶ en la línea de comandos.

Ejemplo

El programa siguiente demuestra la utilización de una variable global para contener los datos utilizados por un programa y los datos de salida que produce. Lleva a cabo las siguientes tareas:

- Acepta un argumento de entrada y calcula su porcentaje con respecto a un valor en la variable global “TOTL”, que usted crea antes de poner en ejecución el programa.
- Almacena el resultado dentro de otra variable global: “RESLT1”.
- Convierte el resultado numérico en una cadena y le agrega “%” para legibilidad.

Para crear el programa, efectúe lo siguiente:

1. Inserte los delimitadores de programa en la línea de comandos y defina la variable de entrada.
   ⮦ « » ⮦ → (ALPHA) A `« → A »`
2. Cree un nuevo procedimiento anidado.
   ⮦ « »
   ```
   « → A
     « »
   »
   ```
3. Introduzca el cálculo de porcentaje.
   ⮌ ( ) (ALPHA) A ÷ (ALPHA) (ALPHA) TOTL (ALPHA) ▶ ⓧ 100
   ```
   « → A
     « (A/TOTL) * 100 »
   »
   ```
4. Almacene los resultados en la variable global "RESLT1". Observe que, después del cálculo, necesitará insertar un ; para delimitar los comandos algebraicos (mantenga pulsada ⮦ y pulse SPC).
   STO▶ (ALPHA) (ALPHA) RESLT1 (ALPHA) ⮦ SPC
   ```
   « → A
    « (A/TOTL) * 100 ▶ RESLT1; »
   »
   ```

5. Agregue “%”, y vuelva a guardar la cadena resultante en RESLT1. Observe lo siguiente:
   – Necesita utilizar la herramienta de caracteres (→ CHARS) para insertar el símbolo %.
   – Al agregar una cadena a un número, el valor resultante es una cadena, por lo que no necesita convertirla.

   ALPHA ALPHA RESLT1 ALPHA
   + " " SPC → CHARS % ENTER ▶ STO▶ ALPHA ALPHA RESLT1

```
« → A B «(A/TOTL)*100 ▶ RESLT1;
RESLT1+" %" ▶ RESLT1»»
```

Antes de poner en ejecución este programa, cree una variable global llamada “TOTL” y asígnele un número.

Bucles y bifurcaciones

Esta sección introduce la utilización de bifurcaciones y bucles condicionales dentro de un programa. Las estructuras condicionales evalúan al 0 como falso, y a cualquier otro valor como verdadero.

Funciones comparativas

La HP 49G brinda funciones de comparación que puede utilizar conjuntamente con las estructuras condicionales y de bucle, pudiendo accederse a las mismas desde el menú *Programming Test* (Prueba de Programación). Por ejemplo, para analizar A en relación a B, utilice lo siguiente:

A==B	Resulta verdadera si A es igual a B.
A≠B	Resulta verdadera si A no es igual a B.
A<B	Resulta verdadera si A es menor que B.
A>B	Resulta verdadera si A es mayor que B.
A≤B	Resulta verdadera si A es menor o igual a B.
A≥B	Resulta verdadera si A es mayor o igual a B.
SAME(A,B)	Resulta verdadera si A es exactamente el mismo objeto que B.

Estructuras condicionales y bucles

Están disponibles los siguientes comandos condicionales y de bucle:

- ***IF*** (SI) *comparación* ***THEN*** (ENTONCES) código ***END*** (FIN)
  Si la *comparación* resulta verdadera, o sea que se trata de un valor distinto de cero, se ejecuta *code* (código).

- **IF** *comparación* **THEN** *código-1* **ELSE** *código-2* **END**

  Si la *comparación* resulta verdadera, ejecuta *código-1*. Si la *comparación* resulta falsa, ejecuta *código-2*.

- **CASE** *expresión-1* **THEN** *código-1* **END**
  expresión-2 **THEN** *código-2* **END**
  . . .
  expresión-n **THEN** *código-n* **END**
  END

  Ejecuta el código correspondiente a la primera expresión en la estructura que resulte verdadera.

- **START (***inicio*, *final***)** *código* **NEXT**

  Ejecuta *código*, incrementa *inicio*. Repite hasta que *inicio*> *final*. El *código* se ejecuta siempre por lo menos una vez.

- **START (***inicio*, *final***)** *código* **STEP** *(incr)*

  Ejecuta *código*, incrementa *inicio* por el número epecificado por *incr*. (*incr* puede ser una expresión.) Repite hasta que *inicio*> *final*. El *código* se ejecuta siempre por lo menos una vez.

- **FOR (***var*, *inicio*, *final***)** *código* **NEXT**

  Ejecuta *código*, asigna *var* a *inicio*. Incrementa *var*, y lo repite hasta que *var* > *final*. Es similar a **START** . . . **NEXT** excepto que usted puede utilizar *var* en su código.

- **FOR (***var*, *inicio*, *final***)** *código* **STEP** *(incr)*

  Ejecuta *código*, incrementa *var* por el número especificado por *incr*. (*incr* puede ser una expresión.) Repite hasta que *inicio*> *final*. Es similar a **START** . . . **STEP** excepto que puede utilizar *var* en su código.

- **DO** *código* **UNTIL** *comparación* **END**

  Ejecuta *código*, luego prueba para ver si *comparación* la evalúa como verdadera. Finaliza si resulta así. Repite *código* si no es así. El *código* siempre se ejecuta por lo menos una vez.

- **WHILE** *comparación* **REPEAT** *código* **END**

  Verifica si *comparación* lo evalúa como verdadera. Ejecuta el *código* si es así. Lo repite hasta que la prueba resulte falsa. Es similar a **DO** . . **UNTIL** excepto que el *código* no se ejecuta si *comparación* evalúa como falsa la primera vez.

Ejemplo

El ejemplo siguiente procesa una lista de valores numéricos almacenados en una variable llamada MARKS. Realiza lo siguiente:

- Determina el número de elementos de la lista.
- El programa compara cada elemento de la lista con el valor a superar:
  a. Si el elemento es mayor o igual que el valor a superar, inserta *"Pass"* (Supera) después del valor.
  b. Si el elemento es menor que el valor a superar, inserta *"Fail"* (Fracasa) después del valor. Observe que esto convierte el valor en una cadena.
- Reemplaza el valor original con la cadena.

```
« @ Utiliza la variable local S
  @ para almacenar el número de pasos.
  @ Pasa de 1 al tamaño de la lista.
  FOR(S,1,SIZE(MARKS))
  @ Extrae el elemento de la lista
    GET (MARKS, S)→E
  @ Lo compara con el valor a superar, enmienda y
  @ reemplaza con el nuevo valor.
    « IF E ≥ 50 THEN
      E + " Pass" → E
     ELSE
      E + " Fail" → E
     END
     REPL ( MARKS, S, E )
     »
  NEXT
  »
```

Cómo detectar errores

Por defecto, un programa se interrumpe cuando encuentra un error. Si desea que algunas secciones del programa se ocupen del error en lugar de interrumpir el programa, debe incluir estas secciones dentro de estructuras de detección de errores. Puede entonces especificar qué acción tomar, en lugar de una interrupción, cuando el programa encuentre errores. Se encuentran disponibles las siguientes estructuras de detección de errores.

- **IFERR** *código* **THEN** *código-error* **END**

  Si el programa encuentra un error mientras se encuentra ejecutando *código*, se saltea el código que resta y se ejecuta *código-error*. Si no se encuentran errores en *código*, no se ejecuta *código-error*.

- **IFERR** *código* **THEN** *código-error* **ELSE** *código-no error* **END**

  Si el programa encuentra un error mientras se encuentra ejecutando *código*, se saltea el código que resta y se ejecuta *código-error*. Si no se encuentran errores en *código*, se ejecuta el *código-no error*.

Ejemplo

El siguiente ejemplo crea una lista de marcas utilizadas en el ejemplo anterior. Si se introduce un valor no numérico, el programa advierte con un mensaje de error. El programa lleva a cabo lo siguiente:

- Establece un bucle para recolectar 20 valores.
- Requiere la entrada de un valor.
- Intenta convertir el valor de una cadena a un número. (Todo lo reunido mientras se ejecuta un programa se encuentra en forma de cadena.)
- Si el intento de conversión fracasa, el error resultante es detectado y aparece un mensaje de error requiriendo un valor numérico.

```
«
@ Creación de una lista vacía
{} ▶ MARKS
@ Establezca un bucle para 20 entradas.
WHILE SIZE(MARKS) < 21 REPEAT
@ Comience con la rutina de verificación de
errores.
  IFERR
    INPUT("Enter a number","")→N
    @ Intenta convertir la entrada a un número.
    @ Esto genera un error si no es numérico.
    « OBJ→ (N) → N
    @ Si no hay error, adjunte lo ingresado a la
lista.
      « MARKS + N ▶ MARKS»
   »
  THEN
    @Si lo ingresado no es numérico aparece lo
siguiente:
    MSGBOX("Invalid entry.Try again.")
  END
END
»
```