Tipos de funciones y procedimientos

Sitio: Agencia de Aprendizaje a lo largo de la Vida
Curso: Tecnicas de Programación 1° F
Libro: Tipos de funciones y procedimientos

Imprimido por: MARIO DAVID GONZALEZ BENITEZ
Día: sábado, 26 de octubre de 2024, 14:58

Tabla de contenidos

1. ¿Cuándo usar parámetros?

Los parámetros son utilizados para transferir información desde el programa principal a una rutina o entre rutinas.

> **Por ejemplo**
>
> Cuando con una calculadora queremos sumar 5 + 7, cada uno de estos valores es un parámetro para la máquina. Son los valores que necesita la calculadora para poder realizar su tarea.

Entonces,

- Un parámetro cumple con el mismo objetivo que una variable: guardar datos o información durante un tiempo dentro del programa o las rutinas.
- Una variable es un elemento que tiene un valor y puede ser modificado en un programa, mientras un parámetro es un valor que se le pasa a una función para que esta pueda realizar su tarea específica.

En la declaración de una rutina (que, recordemos, es lo mismo que método o función), éstas pueden recibir n parámetros, separados por comas (,) e indicando en cada uno de ellos el nombre de la variable que llevará dentro de la rutina.

Conceptualmente, es posible separar los parámetros en dos tipos: los de entrada y los de salida.

> Los parámetros de entrada son aquellos cuyos valores deben ser proporcionados por el programa principal y los parámetros de salida son aquellos cuyos valores se calcularán en la rutina y se deben devolver al programa principal para su proceso posterior.

Cuando se invoca a un módulo se dice que le pasamos argumentos o parámetros reales o actuales. Sin embargo cuando escribimos la rutina, se dice que recibe parámetros formales.

Para nuestro curso:

- llamaremos argumentos al momento de enviar valores a la rutina y
- parámetros al momento de escribir el encabezado de la rutina.

El encabezado de una rutina es la primera línea de código en una función, que especifica el nombre de la rutina y los parámetros que se deben pasar a la rutina cuando se llama. El encabezado también puede especificar el valor que la rutina devuelve.

Observemos que al momento de invocar a la función, utilizamos las variables que tenemos definidas en el programa principal.

```
'El resultado es: ',sumar(numero1,numero2)
```

Sin embargo, nuestro encabezado de la función es:

```
Funcion res = sumar(nro1,nro2)
```

¿Por qué es esto? Porque sumar es una función que sabe realizar una tarea y esa tarea es la de sumar 2 números que reciba como parámetro. Internamente la función conoce los parámetros que recibe que son nro1 y nro2. Estos parámetros formales actúan como variables dentro de la rutina.

Esto nos permite llamar a la función con cualquier variable que tengamos.

¿Cómo funciona? Como dijimos, lo que importa es:

- la cantidad (si pasamos 2 argumentos, debemos recibir 2 parámetros),
- el orden (en este caso, por tratarse de una suma no varía pero si se trata de una división o una resta es importante ya que no es una operación conmutativa),
- tipo de datos (si le paso 2 argumentos enteros, debo recibir 2 parámetros enteros).

Cuando invocamos a un método, el valor del primer argumento se “copia” en el primer parámetro, el valor del segundo argumento se “copia” en el segundo parámetro y así sucesivamente si tuviésemos más argumentos y parámetros.

2. Encabezado de una función

```
Funcion res = sumar(nro1,nro2)
```

Para crear una función, el primer paso es escribir su encabezado, que consta de los siguientes elementos: dato que la función devuelve (en nuestro ejemplo será res), seguido del nombre de la función (elegido por el programador en este ejemplo es sumar) y entre paréntesis el o los parámetros que la función necesita para poder trabajar.

3. Tipos de Funciones

Existen básicamente 4 tipos de funciones:

1. Funciones que no reciben datos ni devuelven un valor producido.

Ejemplo:

Tenemos una función que presenta qué hace el programa y cómo funciona.

Podría ser el caso de un manual para el usuario.

2. Funciones que sólo reciben datos.

Ejemplo:

Tenemos una función que recibe valores y muestra por pantalla dichos valores recibidos por parámetro.

3. Funciones que devuelven un valor.

Ejemplo:

Tenemos una función que le pide al usuario que ingrese un número por teclado y se lo retorna a quien lo haya invocado.

4. Funciones que reciben datos y que también devuelven un valor.

Ejemplo:

Tenemos una función que recibe 2 números y los devuelve multiplicados.

4. Desarrollo de una función

Luego de haber escrito el encabezado de la función, irá el desarrollo de la misma, que consiste en una serie de acciones o instrucciones o llamados a otros módulos, cuya ejecución hará que se asigne un valor al nombre de la variable que vayamos a devolver.

Esto determina el valor particular del resultado que ha de devolverse al programa llamador.

4.1. Procedimientos

Aunque las funciones son herramientas de programación muy útiles para la resolución de problemas, su alcance está muy limitado para la programación estructurada. Con frecuencia se requieren subprogramas que calculen varios resultados en vez de uno solo, o que realicen el ordenamiento de una serie de números, etc. En estas situaciones la función no es apropiada y se necesita disponer del otro tipo de subprograma: el procedimiento.

Un procedimiento, al igual que una función, es un subprograma que ejecuta un proceso específico. Ningún valor está asociado con el nombre del procedimiento; por consiguiente, no se puede usar este nombre en una expresión.

A un procedimiento se lo invoca escribiendo su nombre. Cuando se invoca el procedimiento, cada parámetro formal toma como valor inicial el valor del correspondiente argumento enviado, se ejecutan los pasos o sentencias que definen al procedimiento y a continuación se devuelve el control al programa que lo llamó.

4.2. Utilización de “parámetros” y “argumentos” en procedimientos

Los procedimientos pueden recibir valores de entrada, llamados parámetros. Además, de manera opcional se le puede agregar las palabras claves por valor o por referencia para indicar el tipo de parámetro en cada argumento.

Parámetros por valor

Cada parámetro del procedimiento recibe el “valor” del argumento desde donde es invocado. Sólo recibe “una copia” del valor enviado y si dentro del procedimiento este valor es alterado, el argumento que le corresponde a quien lo invocó, no verá ese cambio.

Parámetros por referencia

Cada parámetro del procedimiento recibe la “referencia” del argumento desde donde es invocado. Recibir “una referencia” implica que tanto el argumento como el parámetro apuntan a la misma variable, con lo cual, si dentro del procedimiento este valor es alterado, el argumento que le corresponde a quien lo invocó, también verá alterado el valor.

5. Procedimiento versus función

Los procedimientos y funciones son subprogramas cuyo diseño y misión son similares; sin embargo, existen unas diferencias esenciales entre ellos.

- Un procedimiento es llamado desde el algoritmo o programa principal mediante su nombre y una serie de argumentos necesarios. Al llamar al procedimiento se detiene momentáneamente el programa que se estuviera realizando y el control pasa al procedimiento llamado. Después que las acciones del procedimiento se ejecutan, se regresa a la acción inmediatamente a la siguiente instrucción a la que se llamó.
- Las funciones devuelven siempre un valor, los procedimientos pueden devolver 0,1 o n valores y en forma de parámetros .
- El procedimiento se declara igual que la función, pero su nombre no está asociado a ninguno de los resultados que obtiene.

Reglas sobre parámetros

Algunas reglas a tener en cuenta para utilizar uno u otro tipo de parámetro son:

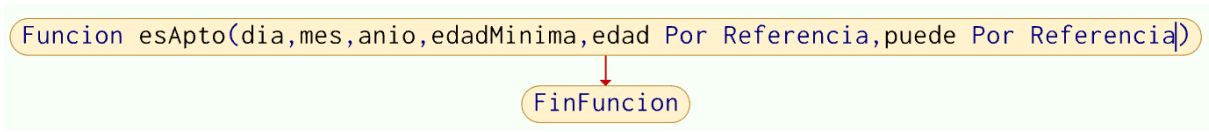
- Si la información que se pasa a la rutina no tiene que ser devuelta fuera de la rutina, el parámetro formal que representa la información puede ser un parámetro por valor (parámetro de entrada).
- Si se tiene que devolver información al programa llamador, el parámetro formal que representa esa información debe ser un parámetro por referencia (parámetro de salida).
- Si la información que se pasa a la rutina puede ser modificada y se devuelve un nuevo valor, el parámetro formal que representa a esa información debe ser un parámetro por dirección (parámetro de entrada/salida).

5.1. Ejemplos de procedimiento vs función

Pongamos como primer ejemplo un procedimiento que le daremos como datos de entrada, la fecha de nacimiento de una persona y la edad mínima para votar en el país.

Este procedimiento nos indicará, a través de dos parámetros de salida, la edad de la persona y si es apto o no para votar.

Este es el encabezado del procedimiento:



Observá que los parámetros de salida en este caso indican que estos parámetros van a devolver un valor al programa principal o la rutina desde donde utilizamos el procedimiento esApto.

La llamada a este procedimiento requiere de 6 argumentos. Es importante notar que los primeros 4 son por valor, de modo que podemos enviarle los datos a través de una variable o bien con datos explícitos, por ejemplo:

Pseudocódigo principal:

```
…. …. …… ……

…. …. …… ……

esApto(14,08,1978, EDAD_MINIMA, edadReal, puede)

…. …. …… ……

…. …. …… ……
```

En este ejemplo, las variables edadReal y puede fueron previamente declaradas y en el momento de la llamada puede que tengan o no un valor asignado, lo cual no es importante ya que ambas variables actúan como parámetro de salida. Esto significa que cuando se termine de ejecutar el procedimiento, estas variables contendrán los datos asignados a los parámetros formales edad y puedeVotar dentro del procedimiento.

EDAD_MINIMA es una constante.

Otro ejemplo válido es utilizando variables que ya contienen datos:

Este otro ejemplo nos muestra cómo utilizar variables para la llamada del procedimiento.

Es importante volver a aclarar que el parámetro día de la llamada es una variable diferente al declarado en el encabezado de la función. Uno es el parámetro actual y otro el formal (ver explicación detallada anteriormente en esta clase).

El diagrama de la llamada al procedimiento se representa de este modo:

esApto(dia,mes,anio,EDAD_MINIMA, edadReal,puede)

Si nos envían una fecha de nacimiento, deberíamos validarla. Para esto, podríamos realizar una función, cuyo encabezado sería el siguiente:

Funcion esFechaValida = fechaValida(dia, mes, anio)

Recordá que, al ser una función, no debe tener parámetros de salida, ya que el resultado se devuelve desde la misma. En este caso, la función debería devolver VERDADERO si la fecha enviada por parámetros es válida y FALSO en caso de que no lo sea.

Veamos este ejemplo para hacer la llamada a la función:

```
/*Declaración de Variables*/

Definir esValida como Logico

Pseudocódigo Principal:

…. …. …… ……

…. …. …… ……

esValida = fechaValida(dia, mes, anio)

…. …. …… ……

…. …. …… ……

…. …. …… ……

…. …. …… ……
```

Como la función devuelve un valor de tipo Lógico, es correcto asignar el resultado de la función a una variable del mismo tipo.

Al ser un valor lógico, también puede utilizarse directamente como condición lógica, por ejemplo:

```
/*Declaración de Variables*/

Pseudocódigo Principal:

…. …. …… ……

…. …. …… ……

Si fechaValida(dia, mes, anio) Entonces

…. …. …… ……

…. …. …… ……

Sino

…. …. …… ……

…. …. …… ……

FinSi
```

Práctica explicada

- Enunciado

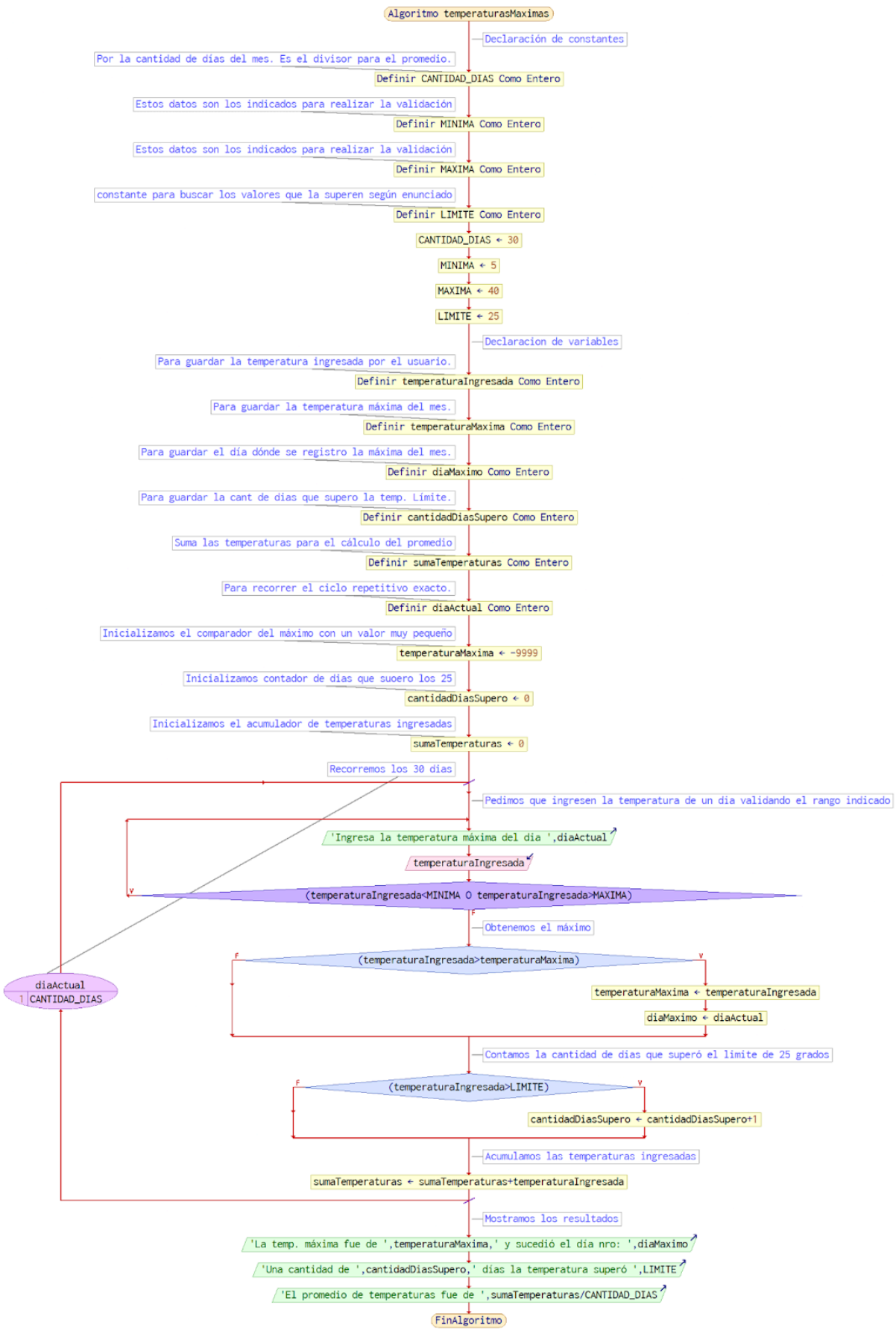
Se leen 30 valores enteros (comprendidos entre 5 y 40) que representan la temperatura máxima de cada de unos de los días de un mes, se pide hallar e informar:

1. La temperatura máxima del mes y el día que se produjo (se supone único).
2. Cuántos días la temperatura superó los 25º.
3. El promedio de las temperaturas máximas del mes.

Hagamos primero un pequeño análisis de qué es lo que debemos hacer para resolver este problema.

- Datos de entrada
  1. Temperatura máxima (valor entero) de un día.
  2. La ingresarán 30 veces (una por cada día del mes).
  3. La temperatura no puede estar por debajo de 5 y por encima de 40 (datos importantes para validar).
- Datos de salida
  1. Temperatura máxima del mes.
  2. Qué día se produjo la temperatura máxima del mes.
  3. Cantidad de días que la temperatura superó los 25º.
  4. El promedio de las temperaturas máximas del mes (es decir, de todas las temperaturas que se ingresen.

Al necesitar buscar un máximo, se debe inicializar alguna variable con un número muy chico, para que el primer dato que me ingresen pase a ser el máximo.



Este ejemplo permite realizar varios procedimientos y, por ejemplo, una función. Vamos a ir deduciendo cuáles son los usos más comunes de los procedimientos.

Las primeras sentencias se refieren a la inicialización de las variables. Es de uso habitual realizar todas las inicializaciones en un módulo aparte.

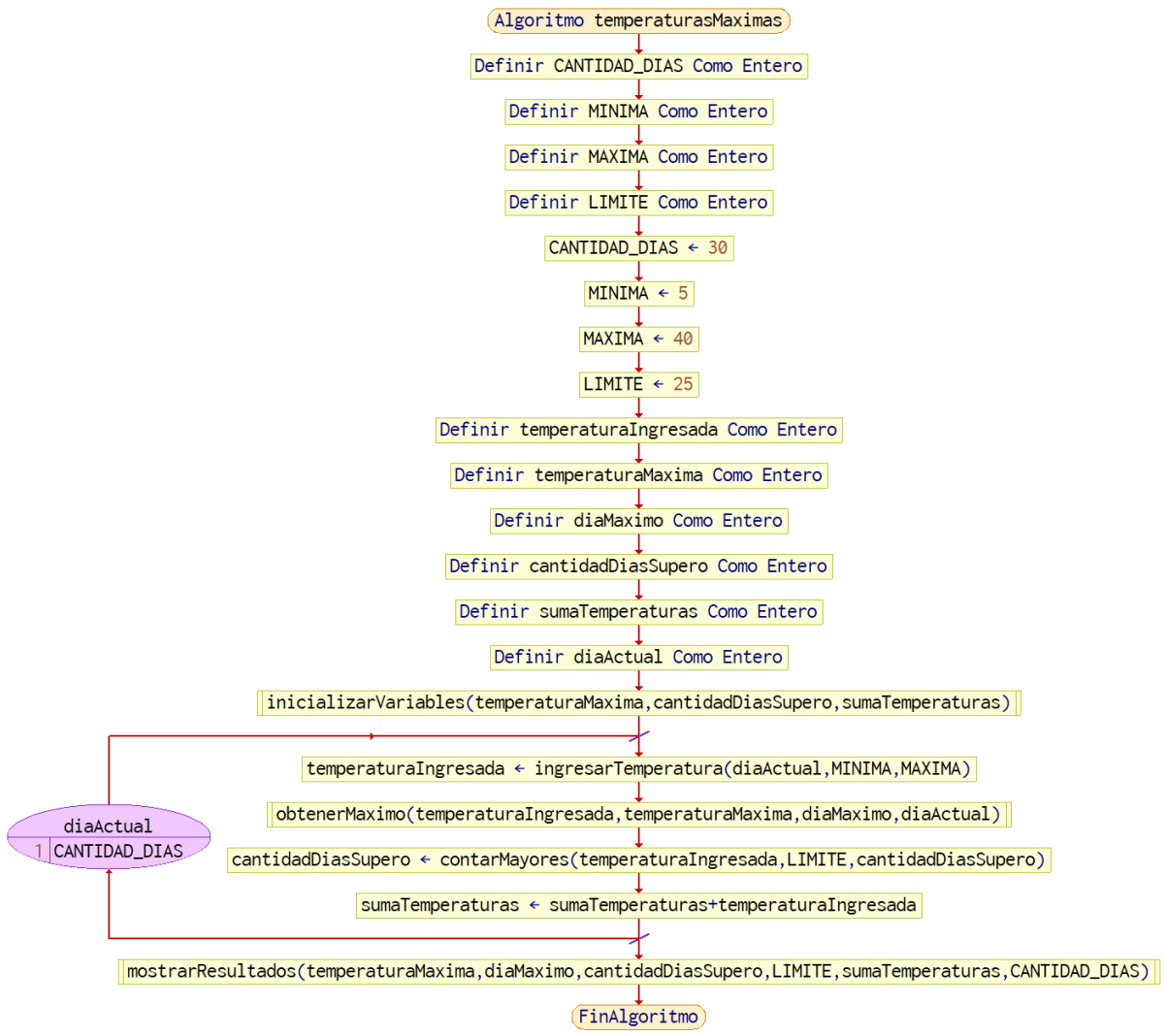
Por otra parte, cada vez que ingresa al ciclo, pide el ingreso de la temperatura. Esto también podría separarse en una función de ingreso de datos.

Los otros dos procedimientos pueden ser los que procesan la información de temperaturaIngresada, para cumplir con los dos primeros objetivos del enunciado.

Finalmente, podríamos realizar una función para el promedio y un procedimiento final para las salidas.

Si fuéramos a utilizar estos módulos que nombramos, el diagrama principal quedaría de este modo:

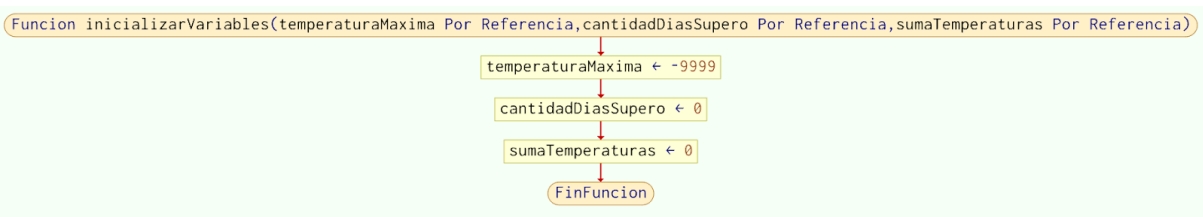
Observemos como. al modularizar, los comentarios dejan de tener sentido ya que el programa principal nos cuenta las tareas que va a ir realizando a medida que se ejecute cada rutina.



Como se ve, no estamos obligados a modularizar exactamente todas las sentencias; en el ejemplo, el acumulador de temperaturas lo dejamos en el programa principal.

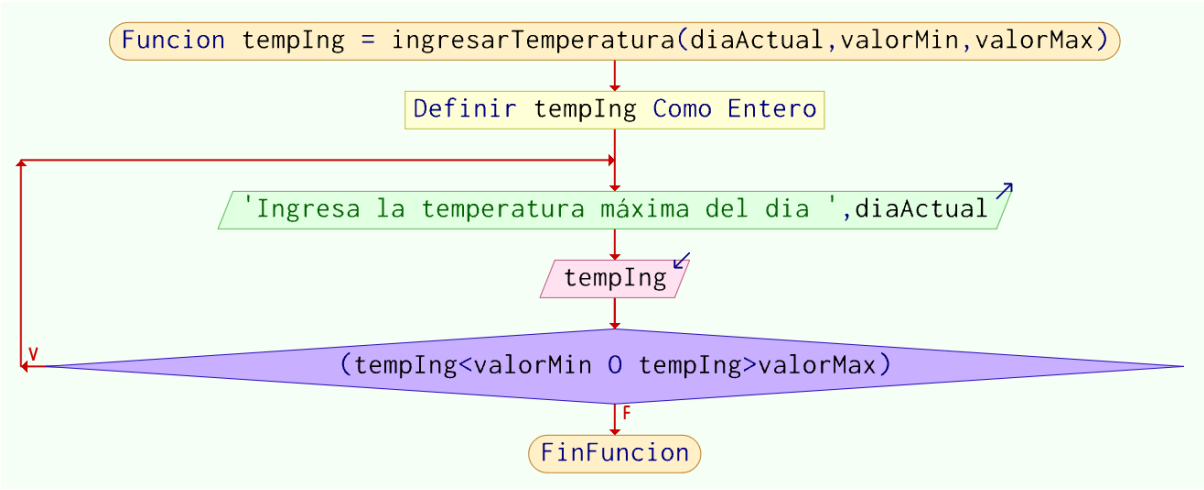
A continuación, desarrollamos cada una de las rutinas. El orden en que se desarrollen es indistinto. Por una cuestión de mejor lectura las ponemos en el orden en que aparecen en el programa principal.

La función inicializarVariables (como mencionamos, técnicamente es un procedimiento ya que devuelve más de un valor modificado) recibe todos sus parámetros por dirección, ya que serán modificados dentro de ésta y el programa principal debe recibir esos cambios.

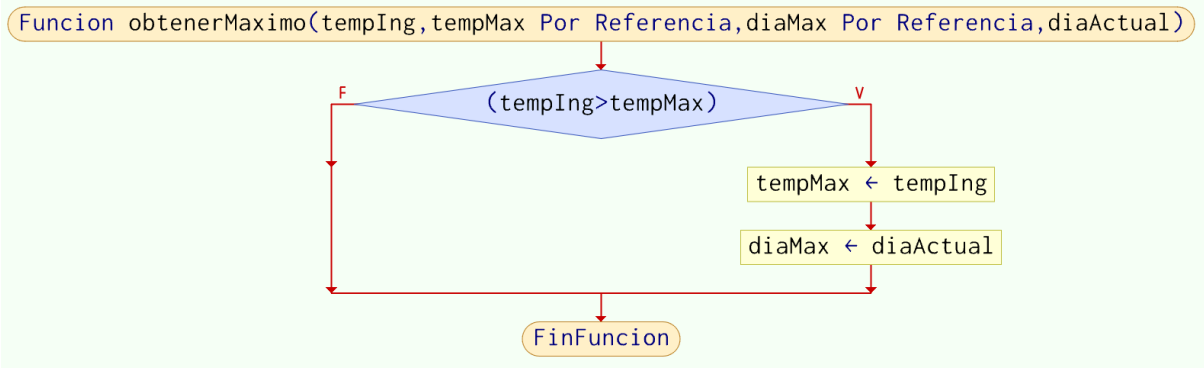


La función ingresarTemperatura, en cambio, sí es una función ya que siempre retornará un único valor que es la temperatura que el usuario ingresó.

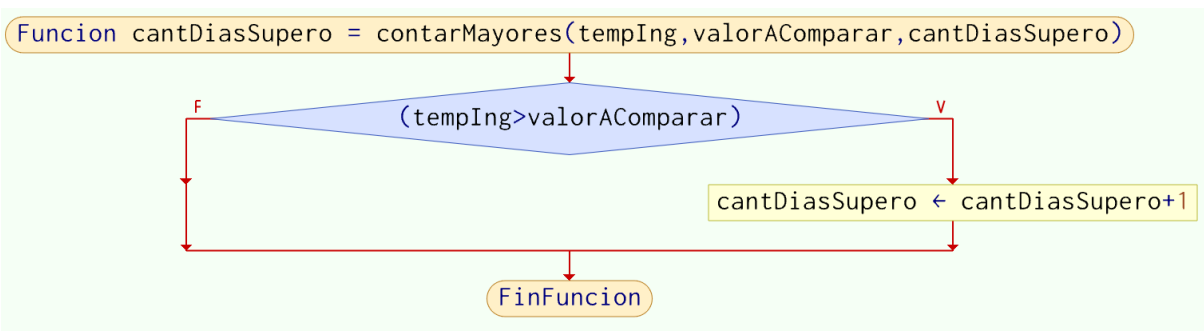
En este caso utilizamos un ciclo Hacer Mientras para validar que el dato ingresado esté dentro de los límites establecidos por el enunciado.



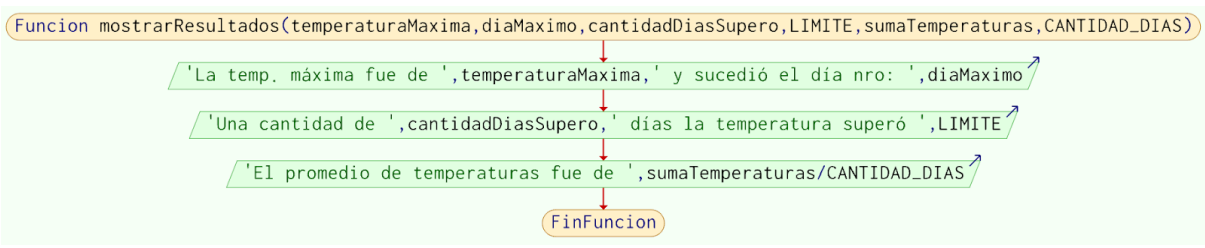
La función obtenerMaximo (es un procedimiento) en caso de detectar que la temperatura ingresada sea mayor a la que teníamos guardada como temperatura máxima, se reemplazarán los valores, pero de no llegar a ser mayor, quedará con los valores que tenía antes de ser invocada. Es decir, podrá devolver 2 valores modificados o bien ninguno de acuerdo a los valores que evalúe la condición.



La rutina contarMayores es una función, ya que siempre devolverá un valor que es la cantidad de veces que la temperatura superó el límite que nos pide el enunciado.



La función mostrarResultados es un procedimiento ya que recibe 3 parámetros pero sólo los debe mostrar por pantalla con lo cual, no se debería modificar ninguno de ellos.



6. Beneficios de la modularización

Los beneficios de la modularización son muchos, aunque cada uno se centra en mejorar la capacidad de mantenimiento y la calidad general de una base de código. En la siguiente tabla, se resumen los beneficios clave.

Beneficios de la modularización

Beneficio	Resumen
Capacidad de reutilización	La modularización ofrece oportunidades para compartir el código. Un programa debe ser una suma de funciones cuando estas se organizan como módulos separados.
Escalabilidad (se desea bajo acoplamiento)	Es una base de código de acoplamiento alto, un solo cambio puede desencadenar una cascada de alteraciones en partes del código aparentemente no relacionadas. Un proyecto modularizado adecuadamente adoptará el principio de separación de problemas y, por lo tanto, limitará el acoplamiento.
Encapsulamiento	El encapsulamiento significa que cada parte de tu código debería tener el menor conocimiento posible sobre otras partes. El código aislado es más fácil de leer y entender. Cada rutina debe encargarse de una tarea específica.
Capacidad de realizar pruebas.	La capacidad de realizar pruebas determina que tan fácil es probar tu código. Un código que se puede probar es aquel en el que las rutinas se pueden probar fácilmente de forma aislada.

En resumen, conceptualmente puede decirse que desarrollar módulos con independencia funcional, variables locales y mínima comunicación externa mediante parámetros es una buena práctica de programación, que favorece la reutilización y el mantenimiento del software.