aprenderaprogramar.com

Módulos genéricos y paso de parámetros por valor (byval) o por referencia (byref). Ejemplos (CU00207A)

Sección: Cursos

Categoría: Curso Bases de la programación Nivel II

Fecha revisión: 2024

Autor: Mario R. Rancel

Resumen: Entrega nº6 del Curso Bases de la programación Nivel II

24

MÓDULOS GENÉRICOS. PARÁMETROS DE ENTRADA. TIPOS TRANSFERENCIA. TIPOS DE PRODUCTO.

Llamaremos módulo genérico a un módulo que requiera la especificación de unos datos de entrada para poder ser llamado. Los datos de entrada son transferidos al módulo de acuerdo a unas normas, participando en uno o varios procesos.

El módulo genérico puede trabajar únicamente con datos transferidos o bien con datos transferidos, variables locales o variables globales según las necesidades. La sintaxis a emplear será:

```
Módulo [Nombre] (Parámetro requerido 1: Tipo,
                 Parámetro requerido 2: Tipo, ...,
                 Parámetro requerido n: Tipo)
    .
    .
    .
FinMódulo
```

La llamada a un módulo genérico ha de incluir un número de datos igual al de parámetros requeridos y además dichos datos han de ser del tipo requerido:

```
Llamar [Nombre] (Dato 1, Dato 2, ..., Dato n)
```

La lista *Dato 1,* Dato *2, etc.* constituye los datos de entrada para el módulo. Si existen datos de salida del módulo (resultados) estos los dispondremos en variables locales o globales, siguiendo las normas habituales con las mismas.

Los datos de entrada pueden ser variables o expresiones de cualquier naturaleza (números, letras, ...) siempre que el tipo coincida con el requerido. Veamos estos conceptos aplicados a casos sencillos.

```
PROGRAMA EJEMPLO 04 [aprenderaprogramar.com]

    Variables
        Enteras: Dato
        Reales: Resultado

    1. Inicio
        2. Mostrar "Bienvenido"
        3. Llamar Raiz(9)
        4. Llamar Raiz(– 144)
        5. Leer Dato
        6. Llamar Raiz(Dato)
    7. Fin

    Módulo Raiz(Numero: Reales)
        1. Numero = ABS(Numero)
        2. Resultado = SQR(Numero)
        3. Mostrar Resultado
    FinMódulo
```

Comentarios: El programa daría lugar a que se mostrara lo siguiente (suponiendo que dato aporta un valor – 100):

```
3
12
10
```

El programa consta de un algoritmo principal y de un único módulo, genérico. A lo largo del algoritmo principal hay diferentes llamadas al módulo genérico. El ahorro de código respecto a lo que sería un programa en que hubiera que escribir repetidamente las instrucciones es evidente.

En el programa intervienen dos variables y un parámetro. Las dos variables son globales, una de tipo entero y otra de tipo real. El parámetro actúa con propiedades similares a las de una variable local aunque no lo es. Se declara entre paréntesis junto al nombre del módulo donde se indica igualmente el tipo. Su misión es servir como elemento genérico que en cada llamada es reemplazado por el dato que se aporta. El parámetro sólo es invocable dentro del módulo para el que se ha definido o módulos subordinados.

Si el dato facilitado no coincide en tipo con el del parámetro se producirá un error por no coincidencia de tipos. Supongamos que realizamos una llamada como Llamar Raiz(“Nueva York”). El resultado es: Error. No coinciden los tipos. Se esperaba para procesar un tipo real y se ha proporcionado un tipo alfanumérico.

De las dos variables declaradas una es exclusivamente utilizada en el módulo Raiz. Se trata de la variable Resultado. Dada esta circunstancia podría haber sido declarada como variable local del módulo, lo cual resulta positivo para la ordenación clara del programa y el ahorro de recursos del ordenador. Sin embargo, declararla como local supone perder accesibilidad a ella desde otras partes del programa. El programador ha de decidir cómo proceder.

El módulo genérico se demuestra útil y claro para invocar procesos que requieren unos datos de entrada. Evita el tener que usar “variables correo” para enviar datos al módulo por lo que es sencillo realizar múltiples llamadas.

Cuando un módulo es invocado y proporcionado un dato de entrada, dicho dato es transferido como reemplazante del parámetro. Si se invoca una expresión libre como puede ser un número, por ejemplo 9 ó – 144 como empleamos en el caso anterior, el funcionamiento equivale a la transferencia del valor correspondiente al parámetro Numero, que funciona como una variable local con valores transitorios hasta la salida del módulo.

Así la invocación Llamar Raiz(– 144) supone:

1º) Transferencia de valor: *Numero = – 144*

2º) Proceso del módulo con:

2.1) *Numero = 144* (valor absoluto)

2.2) Intervención de *Numero* en el cálculo de *Resultado*

Si se invoca una variable como puede ser Dato, que suponemos tiene un valor inicial – 100, el proceso para la invocación Llamar(Dato) supone:

1º) Transferencia de variable: *Numero* ← *Dato*

2º) Proceso del módulo donde *Dato* reemplaza a *Numero:*

2.1) *Dato = 100* (valor absoluto).

2.2) Intervención de *Dato* en el cálculo de *Resultado.*

Se ha realizado una transferencia de variable en reemplazo de un parámetro. Si la variable se manipula en el módulo, por ejemplo se transforma en valor absoluto con Dato = ABS(Dato), cuando termina el módulo la variable almacena el valor manipulado. Si pusiéramos una línea adicional en el algoritmo principal como:

7. Mostrar "El dato procesado ha sido", Dato

El resultado sería El dato procesado ha sido 100, lo cual no es verdad. En este caso, como en muchos otros, nos puede interesar que la variable sea manipulada en el módulo pero conserve su valor inicial cuando se vuelva al algoritmo principal. En definitiva, se trataría de transferir el valor de la variable en vez de a la variable en sí. Para ello bastará con indicárselo así al ordenador, de la siguiente manera:

```
Llamar [Nombre módulo] (Dato 1, Dato 2, ..., Dato n) PorValor
```

La palabra clave PorValor indica que la variable no es transferida, sólo lo es su valor. En nuestro ejemplo: Llamar Raiz(Dato) PorValor

Si se desea, y aunque se considera el defecto, se puede indicar si la llamada es por variable para que quede más claro, usando:

```
Llamar [Nombre módulo] (Dato 1, Dato 2, ..., Dato n) PorVariable
```

En nuestro ejemplo:

Llamar Raiz(Dato) PorVariable

La expresión Llamar Raiz(– 144) PorValor es redundante pero válida.

La expresión Llamar Raiz(– 144) PorVariable es errónea ya que no existe ninguna variable que pueda ser transferida. Cuando PorVariable afecta a un grupo de datos no siendo aplicable a alguno de ellos, dichos datos se procesan por valor ignorándose la especificación.

En la declaración de parámetros se acepta como sintaxis alternativa agrupar los parámetros por tipos, por ejemplo:

Módulo Raiz(a, b, c, d: Enteros, f, g, h: Reales)

Declara necesario suministrar 7 datos para procesar el módulo. Cuatro de ellos enteros y tres reales. Los datos han de proporcionarse siempre en el mismo orden respetando los tipos, por ejemplo:

Llamar Raiz(70, max, min, med, tam1, 43, Temp)

Produce la siguiente transferencia:

```
a ← 70
b ← max
c ← min
d ← med
f ← tam1
g ← 43
h ← temp
```

Las variables max, min y med han de ser enteras y las variables tam1 y temp han de ser reales. Si alguna de ellas tiene un tipo no concordante se producirá un error por no coincidencia de tipos.

La declaración PorValor ó PorVariable afecta al grupo de datos en su totalidad[1]. Si se deseara discriminar dentro de un grupo de datos para que unos fueran transferibles por valor y otros por variable se habrá de especificar uno a uno cada dato dentro del paréntesis de llamada. Si sólo puede tratarse por valor no hace falta especificarlo.

Próxima entrega: CU00208A

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=36&Itemid=60

[1] Con la excepción de que se procesarán por valor aquellos datos que es imposible transferir por variable.