Empleo de módulos y Procedimientos

Procedimientos definidos por el usuario

Según lo que hemos visto hasta ahora, Visual Basic, almacena el código en módulos. Hay tres clases de módulos: formularios (*.frm*), módulos estándar (*.bas*) y clases (*.cls*). Cada módulo puede contener fundamentalmente:

- Declaraciones (constantes, tipos, variables, etc.).
- Procedimientos conducidos por eventos.
- Procedimientos estándar.

La base de una aplicación en Visual Basic la forman sus procedimientos conducidos por eventos. Un *procedimiento conducido por un evento* es el código que se invoca cuando un objeto reconoce que ha ocurrido un determinado evento.
Cuando varios procedimientos conducidos por eventos necesiten ejecutar un mismo proceso, por ejemplo visualizar un diagrama de barras, la mejor forma de proceder es colocar el código común en un *procedimiento general*, perteneciente a un módulo estándar, para que pueda invocarse desde cada procedimiento que lo necesite. De esta forma se elimina la necesidad de duplicar código. Un *procedimiento* se invoca haciendo una llamada explícita al mismo.
Un procedimiento general puede escribirse como procedimiento *Sub* o como función *Function*. En cambio, un procedimiento conducido por un evento siempre es un procedimiento *Sub*. El nombre de un procedimiento general lo elige el usuario, y el nombre de un procedimiento conducido por eventos lo forma Visual Basic concatenando el nombre del objeto (*Form* o nombre del control) y el nombre del evento; esto es, *objeto_evento*. Aparte de estas diferencias y cómo y cuando son invocados, los procedimientos generales y los conducidos por eventos se implementan y se ejecutan exactamente igual.

Ámbito de un procedimiento

Cuando un procedimiento no se califica explícitamente con las palabras reservadas *Public* o *Private* es, por omisión, *público* en todos los módulos. Lógicamente el carácter *público* de un procedimiento puede ser cambiado a *privado*, modificando así su accesibilidad. Por ejemplo, el siguiente procedimiento escrito en cualquier módulo es público:

```
Sub MiProc()
'Declaraciones y sentencias
End Sub
```

Consecuentemente, un procedimiento público (*Public*) puede invocarse desde cualquier parte de la aplicación, pero un procedimiento privado (*Private*) sólo puede ser llamado desde otros procedimientos que estén en el mismo módulo.
Cuando un procedimiento es llamado para su ejecución, Visual Basic busca ese procedimiento en el módulo donde nos encontremos. Si no lo encuentra, entonces continúa la búsqueda en el resto de los módulos de la aplicación.

Crear un procedimiento general

Para crear un procedimiento general, primero diríjase a la ventana del proyecto, elija el módulo donde quiere definir el procedimiento y abra la ventana de código correspondiente haciendo clic en el botón *Ver código*. A continuación elija la opción *Agregar procedimiento...* del menú *Herramientas*. También, puede escribir *Sub* o *Function* seguido del nombre del procedimiento directamente sobre la ventana de código; en este último caso, al pulsar la tecla *Entrar* Visual Basic completará la esqueleto del procedimiento.
Para editar un procedimiento general existente, seleccione "(*General*)" en la lista *objeto* de la ventana de código y a continuación seleccione el procedimiento en la lista *procedimiento*.

Funciones (Function)

Una función es un procedimiento que cuando se ejecuta devuelve un resultado. La sintaxis correspondiente a una función es la siguiente:

```
[Private|Public] [Static] Function nombre [(parámetros)] [As tipo]
  [sentencias]
  [nombre = expresion]
  [Exit Function]
  [sentencias]
  [nombre = expresión]
End Function
```

nombre es el nombre de la función; su tipo determina el tipo de datos que devuelve la función. Para especificar el tipo, utilice los caracteres de declaración de tipo o la cláusula *As tipo* (*Integer*, *Long*, *Single*, *Double*, *Currency*, *String* (sólo de longitud variable), *Variant*, *tipo definido por el usuario*, etc.); el tipo es *Variant* por omisión.

parámetros son una lista de variables separadas por comas que se corresponden con los argumentos que son pasados cuando es invocada la función. Cuando se llama a una función, Visual Basic asigna el valor de cada argumento en la llamada al parámetro que ocupa su misma posición en la lista de parámetros de la función.

expresión define el valor devuelto por la función. Este valor es almacenado en el propio *nombre* de la función, que actúa como variable dentro del cuerpo de la misma. Si no se efectúa esta asignación, el resultado devuelto será 0 si ésta es numérica, nulo (“”) si la función es de caracteres, o vacío (*Empty*). si la función es *Variant*.

Exit Function permite salir de una función. *Exit Function* no es necesaria a no ser que se necesite retornar a la sentencia situada inmediatamente a continuación de la que efectuó la llamada antes que la función finalice.

End Function esta sentencia, al igual que *Exit Function*, devuelve el control a la sentencia que se encuentra inmediatamente a continuación de la que efectuó la llamada, continuando de esta forma la ejecución del programa.

La llamada a una función es de la forma

[variable =] nombre [(argumentos)]

argumentos son una lista de constantes, variables o expresiones separadas por comas. El número de argumentos debe ser igual al número de parámetros de la función. Los tipos de los argumentos deben coincidir con los tipos de sus correspondientes parámetros.

Por compatibilidad con otros lenguajes, es aconsejable escribir los paréntesis cada vez que se llama a una función, aunque ésta no tenga argumentos. Una llamada a una función puede formar parte de una expresión.

Procedimiento Sub

La sintaxis que define un procedimiento es la siguiente:

```
[Private|Public] [Static] Sub nombre[(parámetros)]
  [sentencias]
  [Exit Sub]
  [sentencias]
End Sub
```

La explicación es análoga a la dada para las funciones(*Function*).
La llamada a un procedimiento puede ser de alguna de las dos formas siguientes:

```
Call nombre [(argumentos)]
nombre [argumentos]
```

A diferencia de una función, un procedimiento no puede ser utilizado en una expresión, ya que un procedimiento no retorna un valor a través de su nombre.
El siguiente ejemplo corresponde a un procedimiento (*Sub*) que calcula e imprime la suma de los elementos de una matriz.

Llamar a procedimientos en otros módulos

Para llamar a un procedimiento público (*Sub* o *Function*) de un formulario desde cualquier otro módulo, hay que utilizar la siguiente sintaxis:

form.procedimiento(argumentos)

donde *form* representa el formulario al cual pertenece el procedimiento llamado. Por ejemplo:

```
Public Sub procSuma(x() As Double, n As Integer)
  Dim i, Suma As Single
  'Calcular la suma de los valores de la matriz x
  For i = 1 To n
    Suma = Suma + x(i) 'acumulador
  Next i
  frmFunProc.Print "El pago total de la empresa es "; Suma
End Sub
```

En este caso se trata de un proceso predefinido, pero la regla se aplica exactamente igual a los procedimientos definidos por el usuario.
Para llamar a un procedimiento público de un módulo estándar desde cualquier otro módulo, puede hacerlo de las dos maneras siguientes:

módulo.procedimiento(argumentos)
procedimiento(argumentos)

donde *módulo* se refiere al nombre del módulo estándar al que pertenece el procedimiento. El nombre del módulo será obligatorio especificarlo cuando existan dos o más procedimientos con el mismo nombre en diferentes módulos.

Declarar todas las variables locales como estáticas

Para hacer que todas las variables locales de un procedimiento (*Sub* o *Function*) sean por omisión estáticas, hay que colocar al principio de la cabecera del procedimiento la palabra clave Static. Por ejemplo:

```
Public Static Sub Pro_1(X As Double, N As Integer)
  '...
End Sub
```

Declarar un procedimiento privado

Para hacer que un procedimiento (*Sub* o *Function*) sólo sea accesible desde los procedimientos del módulo al cual pertenece, hay que colocar al principio de la cabecera del procedimiento la palabra clave *Private*. Por ejemplo:

```
Private Sub Proc_1(X As Double, N As Integer) .
  '...
End Sub
```

Si no se especifica la palabra clave *Private* se supone que el procedimiento es *Public*, lo que significa que puede ser invocado desde otros módulos.

Argumentos por referencia y por valor

En los procedimientos (*Sub* o *Function*), los argumentos se pasan por referencia; de este modo, cualquier cambio de valor que sufra un parámetro en el cuerpo del procedimiento, también se produce en el argumento correspondiente de la llamada al procedimiento. Por ejemplo, en la función *fnPromedio*, el argumento es pasado por referencia.

```
promedio = fnPromedio(pagos) 'se pasa como argumento la matriz
                                'pagos

Public Function fnPromedio(dinero As Double) As Double
'...
End Function
```

Esto quiere decir que si en el cuerpo de la función *fnPromedio* se modificara el valor de la variable *dinero*, dichos cambios también serían vistos desde *pagos*, porque pasar *pagos* por referencia significa que ambas variables, referencian los mismos valores. Cuando se pasa un parámetro por referencia, lo que realmente se pasa al procedimiento es dónde están los datos con los que tiene que trabajar, no una copia de los datos.
Cuando se llama a un procedimiento (*Sub* o *Function*), se podrá especificar que el valor de un argumento no sea cambiado por ese procedimiento, poniendo dicho argumento entre paréntesis en la llamada. Un argumento entre paréntesis en la llamada es un argumento pasado por valor. Por ejemplo:

```
procSuma pagos, (nroEmpleados)

Public Sub procSuma(dinero As Double, n As Integer)
  '...
End Sub
```

Observe la llamada al procedimiento *procSuma*; el argumento *nroEmpleados* es pasado por valor, lo cual significa que se pasa una copia de *nroEmpleados* (el argumento *nroEmpleados* en la llamada y el parámetro n del procedimiento no se confunden porque los parámetros de un procedimiento son locales al mismo). Por lo tanto, si el procedimiento *procSuma* cambiara el valor de *n*, el cambio afecta sólo a este procedimiento y no al argumento *n* en la llamada; por esto decimos que cuando se pasa un argumento por valor se pasa una copia, impidiendo así trabajar sobre la variable original.
Otra forma de especificar que un argumento será pasado por valor es anteponiendo la palabra *ByVal* a la declaración del parámetro en la cabecera del procedimiento (*Sub* o *Function*). Análogamente, *ByRef* especifica que el parámetro será pasado por referencia; por omisión se supone *ByRef*. Por ejemplo:

```
Public Sub procSuma(dinero As Double, ByVal n As Integer).,
```

La cabecera del procedimiento *procSuma* especifica que *dinero* será pasado por referencia y que *n* será pasado por valor. Esta forma de proceder evita tener que poner *n* entre paréntesis en la llamada.
Una estructura (tipo definido por el usuario) también se puede pasar como argumento a un procedimiento. Así mismo, una función puede retornar una estructura. Pero en el caso de un formulario o de una clase, el procedimiento (*Sub* o *Function*) tiene que ser *Private*. Los argumentos de un tipo definido por el usuario son siempre pasados por referencia.

Argumentos opcionales

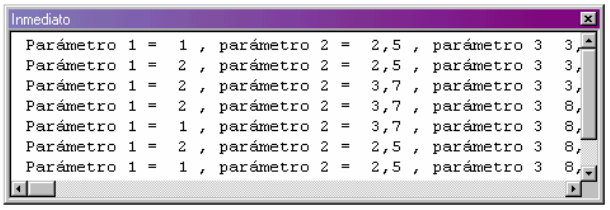
La lista de parámetros de un procedimiento puede incluir parámetros opcionales utilizando la palabra clave *Optional*. Si se especifica un argumento opcional, todos los argumentos subsiguientes de la lista de argumentos deben ser también opcionales y se deben declarar con la palabra clave *Optional*.
También es posible especificar un valor predeterminado para un argumento opcional. Si no se especifica, tomará el valor predeterminado por Visual Basic (0 para los parámetros numéricos, *Empty* para los *Variant*, etc.). Por ejemplo, el siguiente procedimiento proporciona todos los argumentos como opcionales:

```
Public Sub Visualizar(Optional a As Integer = 1, _
                      Optional b As Single = 2.5, _
                      Optional c As Double = 3.456)
  Debug.Print "Parámetro 1 = "; a;
  Debug.Print ", parámetro 2 = "; b;
  Debug.Print ", parámetro 3 "; c
End Sub
```

Este procedimiento puede ser invocado de las formas siguientes:

```
Private Sub Commandl_Click()
  Visualizar
  Visualizar 2
  Visualizar 2, 3.7
  Visualizar 2, 3.7, 8.125
  Visualizar , 3.7, 8.125
  Visualizar 2, , 8.125
  Visualizar , , 8.125
End Sub
```

Cuando ejecute este código, en la ventana Inmediato se visualizarán los resultados siguientes:

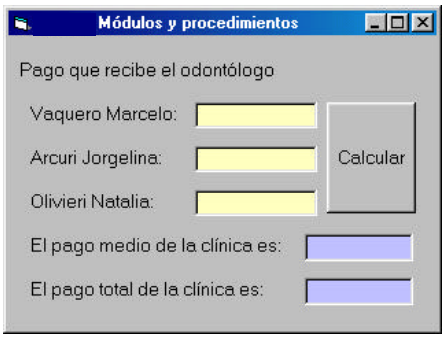


Sin embargo, esta otra versión no proporciona todos los parámetros como opcionales:

```
Public Sub Visualizar(a As Integer, _
                      Optional b As Single = 2.5, _
                      Optional c As Double = 3.456)
  Debug.Print "Parámetro 1 = "; a;
  Debug.Print ", parámetro 2 = "; b;
  Debug.Print ", parámetro 3 "; c
End Sub
```

Observe en este ejemplo que el primer parámetro no está definido como opcional, por lo que siempre habrá que especificar al menos un argumento en la llamada.

Como ejemplo de lo expuesto realizaremos una aplicación simple que calcule el pago promedio de la clínica odontológica *Smile* y del pago total que la misma realiza destinada al pago de los sueldos de sus tres únicos odontólogos. Comience por diseñar el formulario:



A continuación se detallan los controles utilizados junto con los valores de sus propiedades:

Objeto	Propiedad	Valor
Text1	Nombre	txtOdontologo1
Text2	Nombre	txtOdontologo2
Text3	Nombre	txtOdontologo3
Label1	Nombre	lblPagoMedio
Label2	Nombre	lblPagoTotal
Command1	Nombre Caption	cmdCalcular Calcular

Ahora agregue un módulo al proyecto eligiendo la opción *Agregar módulo* del menú *Proyecto*. En el módulo agregado al proyecto se declarará un procedimiento llamado *procSuma* y una función llamada *fnPromedio*. El procedimiento *procSuma* poseerá tres argumentos, todos de tipo *Double* , llamados *imp1*, *imp2* e *imp3*. Dentro del mismo se efectuará la suma de los tres importe y mostrará el resulta en la etiqueta *lblPagoTotal* del formulario. La declaración del procedimiento es la siguiente;

```
Public Sub procSuma(imp1 As Double, imp2 As Double, imp3 As Double)
  Dim Suma As Single
  'Calcular la suma de los valores ingresados en las cajas de texto
  Suma = imp1 + imp2 + imp3
  Form1.lblPagoTotal.Caption = Suma
End Sub
```

El procedimiento *fnPromedio* poseerá tres argumentos, todos de tipo *Double* , llamados *imp1*, *imp2* e *imp3* y devolverá un *Double* que será el pago medio de la clínica. Dentro de la misma se efectuará la suma de los tres importes y se dividirá por tres con el fin de calcular el promedio. La declaración de la función es la siguiente;

```
Public Function fnPromedio(imp1 As Double, imp2 As Double, imp3 _
                           As Double) As Double
  Dim Suma As Single
  'Calcular la media de los valores ingresados
  'en las cajas de texto
  Suma = imp1 + imp2 + imp3
  fnPromedio = Suma / 3 'valor que se devuelve
End Function
```

El procedimiento *procSuma* y la función *fnPromedio* serán invocadas desde el evento clic del botón de pulsación *cmdCalcular*. El código correspondiente a dicho evento es el siguiente:

```
Private Sub cmdCalcular_Click()
  Dim promedio As Double
  promedio = fnPromedio(txtOdontologo1.Text, txtOdontologo2.Text,_
          txtOdontologo3.Text) 'se pasa como argumento los valores
  'ingresados en las cajas de texto
  'Escribir resultados
  lblPagoMedio.Caption = Format(promedio, "#0.00")
  procSuma txtOdontologo1.Text, txtOdontologo2.Text, _
     txtOdontologo3.Text 'se pasa como argumento los valores
  'ingresados en las cajas de texto. También se podría
  'escribir: Call procSuma(pagos, nroEmpleados)
End Sub
```

Al hacer clic sobre el botón *Calcular*, se llamará al procedimiento *procSuma* y a la función *fnPromedio*, esto hará que se muestre el resultado de tales operaciones en las etiquetas correspondientes. Podrá observar que la función retorna un valor, el cual es asignado a la propiedad *Caption* de la etiqueta *lblPagoMedio*.