Procedimientos

Todo el código ejecutable de una aplicación se ubica en rutinas de código o procedimientos. Un procedimiento es un elemento del lenguaje compuesto por un conjunto de líneas de código, a las que se denomina cuerpo del procedimiento. Su comienzo y fin lo establecemos mediante ciertas palabras reservadas del lenguaje, asociándole un identificador, que nos servirá para reconocerlo entre el resto de procedimientos creados en el programa. Podemos enviarle también información adicional en forma de parámetros, con lo que el resultado de la ejecución de un procedimiento variará según los valores que pasemos en cada llamada.

En VB.NET disponemos de los siguientes tipos de procedimientos:

• Sub. Procedimiento que realiza un conjunto de operaciones pero no devuelve valor al punto de llamada. A lo largo del texto también nos referiremos a las rutinas de tipo Sub con el nombre genérico de procedimiento.

• Function. Procedimiento que realiza un conjunto de operaciones, y devuelve un valor denominado valor de retorno al punto de código que realizó la llamada. A lo largo del texto también nos referiremos a las rutinas de tipo Function con el nombre genérico de función.

• Property. Procedimiento que se utiliza para labores de acceso y asignación de valores a las propiedades de un objeto. Serán tratados con más profundidad en el tema dedicado a la programación orientada a objetos.

Sintaxis de un procedimiento Sub

El formato para la escritura de un procedimiento Sub se muestra en el Código fuente 117.

```
[Ámbito] Sub NombreProcedimiento[(ListaParámetros)]
       [CódigoEjecutable]
       [Exit Sub | Return]
       [CódigoEjecutable]
End Sub
```

Código fuente 117

Los elementos que forman parte de este tipo de rutina son los siguientes:

• Ámbito. Define el modo en que vamos a poder acceder o llamar al procedimiento desde otro punto de la aplicación. El ámbito de los elementos del lenguaje será tratado en un apartado posterior.

• Sub...End Sub. Palabras clave que indican el comienzo y final del procedimiento respectivamente. Cuando hagamos una llamada al procedimiento, el compilador ejecutará el código comprendido entre estas dos palabras clave.

• NombreProcedimiento. Identificador que utilizamos para reconocer y llamar al procedimiento.

• ListaParámetros. Lista de variables separadas por comas, y encerradas entre paréntesis, que representan la información que recibe el procedimiento desde el código llamador.

• Return. Esta palabra clave permite salir de la ejecución del procedimiento sin haber llegado a su fin. Podemos utilizarla en tantos lugares dentro de un procedimiento como sea necesario. Se recomienda su uso en lugar de Exit Sub, ya que podemos emplear Return para salir de cualquier tipo de procedimiento, con lo cual se unifica la escritura del código.

• Exit Sub. Al igual que en el punto anterior, esta palabra clave permite salir de la ejecución del procedimiento sin haber llegado a su fin, pudiendo igualmente, situarla en tantos lugares dentro del procedimiento como sea necesario.

El Código fuente 118 muestra el modo más simple de crear un procedimiento. Escriba el lector este procedimiento en la aplicación de consola sobre la que está realizando las pruebas, a continuación de Main( ).

```
Sub Prueba()
    Console.WriteLine("Estamos en el procedimiento Prueba")
End Sub
```

Código fuente 118

Llamada a un procedimiento Sub

Para realizar una llamada o ejecutar un procedimiento Sub, debemos escribir su nombre en un punto del programa. El Código fuente 119 muestra el código al completo del módulo de nuestra aplicación de consola. La ejecución de este programa comienza como es habitual por Main( ), dentro del cual se realiza una llamada al procedimiento Prueba( ).

```
Module Module1

    Sub Main()
        Console.WriteLine("Estamos en el procedimiento Main")

        ' llamada a un procedimiento
        Prueba()

        Console.ReadLine()
    End Sub

    Sub Prueba()
        Console.WriteLine("Estamos en el procedimiento Prueba")
    End Sub

End Module
```

Código fuente 119

En la llamada a un procedimiento Sub, el uso de los paréntesis es opcional, independientemente de si pasamos o no parámetros. No obstante, es muy recomendable especificar dichos paréntesis, ya que aportan una gran claridad a nuestro código, de forma que al leerlo podemos ver rápidamente los puntos en los que se realiza una llamada a una rutina de código. Debido a esto, el IDE sitúa automáticamente los paréntesis en el caso de que no los especifiquemos de forma explícita. No es posible situar la llamada a un procedimiento Sub como parte de una expresión, puesto que este tipo de procedimientos, al no devolver un valor, provocaría un error de compilación. Ver Figura 176.

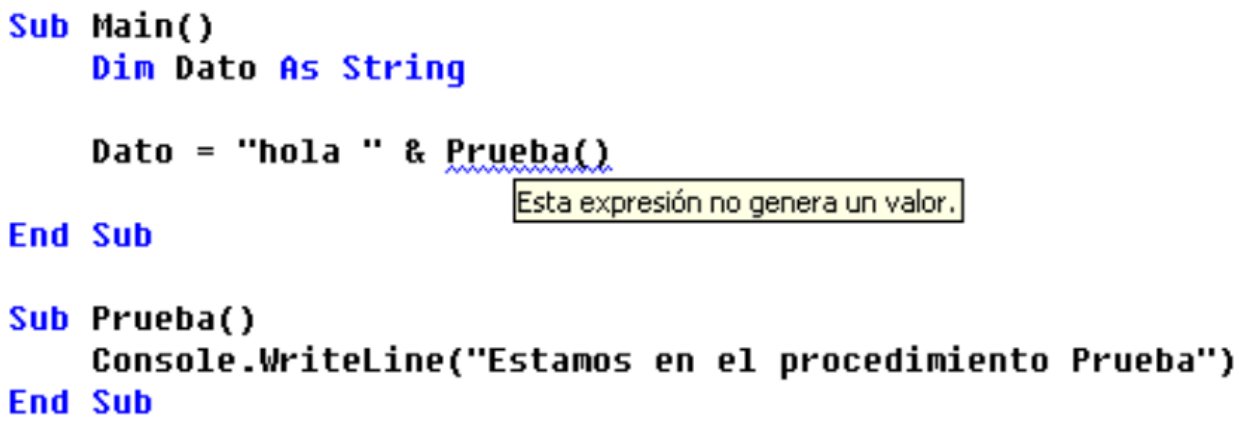

Figura 176. No es posible hacer una llamada a un procedimiento Sub en una expresión.

Sintaxis de un procedimiento Function

El formato para la escritura de un procedimiento Function se muestra en el Código fuente 120.

```
[Ámbito] Function NombreFunción[(ListaParámetros)] As TipoDato
       [CódigoEjecutable]
       [Return Valor]
       [NombreFunción = Valor]
       [Exit Function]
       [CódigoEjecutable]
End Function
```

Código fuente 120

Los elementos que forman parte de este tipo de rutina son los siguientes:

• Ámbito. Define el modo en que vamos a poder acceder o llamar al procedimiento desde otro punto de la aplicación. El ámbito de los elementos del lenguaje será tratado en un apartado posterior.

• Function...End Function. Palabras clave que indican el comienzo y final de la función respectivamente. Cuando hagamos una llamada a la función, el compilador ejecutará el código comprendido entre estas dos palabras clave.

• NombreFunción. Identificador que utilizamos para reconocer y llamar a la función. En este tipo de procedimiento, también utilizamos su nombre para asignar el valor que será devuelto al código llamador en el modo NombreFunción = Valor, en esta última situación, podemos situar esta expresión de devolución en tantos lugares como necesitemos dentro de la función.

• TipoDato. Tipo de dato del valor devuelto como resultado de la ejecución de la función.

• ListaParámetros. Lista de variables separadas por comas, y encerradas entre paréntesis, que representan la información que recibe la función desde el código llamador.

• Return. Esta palabra clave permite salir de la ejecución de la función devolviendo al mismo tiempo un valor al código que hizo la llamada. Podemos utilizarla dentro de una función, en tantos lugares como necesitemos.

• Exit Function. Esta palabra clave permite salir de la ejecución de la función sin haber llegado a su fin. Podemos utilizarla dentro de una función, en tantos lugares como necesitemos.

El Código fuente 121 muestra un sencillo ejemplo de procedimiento Function, en el cual se pide al usuario que introduzca un número que es devuelto como resultado de la función.

```
Function Calcular() As Integer
    Dim MiValor As Integer
    Console.WriteLine("Introducir un número de 1 a 100")
    MiValor = Console.ReadLine()
    Return MiValor
    ' también podemos utilizar esta
    ' sintaxis para devolver el valor
    ' de retorno de la función:
    'Calcular = MiValor
End Function
```

Código fuente 121

En el caso de devolver el valor de retorno de una función utilizando el propio nombre de la función, nos encontramos con el problema de que si en un momento determinado tenemos que cambiar el nombre de la función, también deberemos cambiar todos aquellos puntos de la rutina en donde devolvemos el valor. Por este motivo es recomendable el uso de Return para el devolver el valor de la función, ya que si tenemos que cambiar el nombre de la función, no será necesario modificar los puntos en los que se devuelve el valor de este tipo de procedimiento.

Llamada a un procedimiento Function

Para realizar una llamada o ejecutar un procedimiento Function debemos escribir su nombre en un punto del programa; en este aspecto ambos tipos de procedimiento son iguales. Por otro lado, los puntos que marcan las diferencias entre un Function y un Sub son los siguientes:

• Un procedimiento Function devuelve un valor, de modo que si queremos obtenerlo, debemos asignar la llamada a la función a una variable. Los procedimientos Sub no pueden devolver valores.

• Debido precisamente a la capacidad de un procedimiento Function de devolver un valor, podemos situar la llamada a una función dentro de una expresión, y operar con el valor de retorno dentro de la expresión, lo cual dota a nuestro código de una mayor flexibilidad. Los procedimientos Sub no pueden formar parte de expresiones.

En el Código fuente 122 vemos varios ejemplos de llamadas a la función Calcular( ), según el modo en que vamos a manipular su valor de retorno.

```
Module Module1

    Sub Main()
        Dim Resultado As Integer
        Dim NuevoValor As Integer

        ' llamada a una función sin recoger el valor de retorno,
        ' por este motivo, dicho valor se pierde
        Calcular()

        ' llamada a una función obteniendo el valor
        ' de retorno y asignando el valor a una variable
        Resultado = Calcular()
        Console.WriteLine("La variable Resultado contiene: {0}", Resultado)

        ' llamada a una función como parte de una expresión
        NuevoValor = 1500 + Calcular() * 2
        Console.WriteLine("La variable NuevoValor contiene: {0}", NuevoValor)

        Console.ReadLine()
    End Sub

    Function Calcular() As Integer
        Dim MiValor As Integer

        Console.WriteLine("Introducir un número de 1 a 100")
        MiValor = Console.ReadLine()

        Return MiValor

        ' también podemos utilizar esta
        ' sintaxis para devolver el valor
        ' de retorno de la función:
        'Calcular = MiValor
    End Function

End Module
```

Código fuente 122

Paso de parámetros a procedimientos

Un parámetro consiste en un valor que es pasado a un procedimiento. Dicho valor puede ser una variable, constante o expresión. Los procedimientos pueden recibir parámetros en forma

de lista de variables separada por comas, siguiendo la sintaxis que vemos en el Código fuente 123.

```
[Optional] [ByVal | ByRef] [ParamArray] NombreParametro As TipoDato
```

Código fuente 123

Las reglas y cuestiones sobre paso de parámetros descritas en los siguientes apartados son válidas tanto para procedimientos como para funciones, excepto en aquellos lugares donde se indique lo contrario.

Tipo de dato de un parámetro

Al igual que hacemos cuando declaramos una variable, al declarar un parámetro debemos especificar el tipo de dato que el parámetro va a contener. Esto será o no obligatorio, en función del modificador establecido en la instrucción Option Strict.

En el Código fuente 124 hemos añadido un parámetro de tipo String al procedimiento Prueba( ). Cuando llamemos desde otro procedimiento a Prueba( ), al pasar desde el código llamador una cadena de caracteres, bien de forma literal o en una variable; el contenido de dicha cadena se traspasará a la variable situada en la lista de parámetros del procedimiento, que posteriormente visualizaremos en la consola.

```
Sub Main()
    Dim Nombre As String

    Nombre = "Juan"
    Prueba(Nombre)

    Prueba("Esto es una prueba")
    Console.ReadLine()
End Sub

Sub Prueba(ByVal ValorMostrar As String)
    Console.WriteLine("El valor del parámetro pasado es {0}", ValorMostrar)
End Sub
```

Código fuente 124

Paso de parámetros por valor y por referencia

Existen dos modos en el lenguaje de pasar parámetros a un procedimiento: por valor y por referencia. Este aspecto del lenguaje está relacionado con el modo en que el contenido de los parámetros son gestionados internamente en memoria. Ello nos permitirá según el tipo de paso empleado, poder alterar el valor del parámetro en el código que realizó la llamada.

Paso por valor (ByVal)

Cuando pasamos un parámetro por valor a un procedimiento, la variable que contiene el parámetro puede ser modificada dentro del procedimiento, sin que estos cambios afecten al valor original en el código llamador. Debemos situar en este caso en el procedimiento, la palabra clave ByVal antes del nombre del parámetro. Ver Código fuente 125.

```
Sub Main()
    Dim Nombre As String

    Nombre = "Juan"

    ' llamamos a un procedimiento
    ' y le pasamos una variable por valor
    Prueba(Nombre)

    ' la variable que hemos pasado al procedimiento,
    ' al volver aquí no ha sido cambiada, debido a que
    ' ha sido pasada por valor, sigue conteniendo
    ' la cadena "Juan"
    Console.WriteLine("Valor de la variable dentro de Main(): {0}", Nombre)
    Console.ReadLine()
End Sub

Sub Prueba(ByVal ValorMostrar As String)
    ' modificamos el valor del parámetro,
    ' este cambio no afecta a la variable Nombre
    ValorMostrar = "Elena"
    Console.WriteLine("Valor del parámetro dentro de Prueba(): {0}", ValorMostrar)
End Sub
```

Código fuente 125

Lo que ocurre en el fuente anterior a nivel de gestión interna en memoria de los parámetros es lo siguiente: cuando se realiza la llamada al procedimiento y se pasa el parámetro, el entorno detecta que se trata de un parámetro pasado por valor, por lo que crea una nueva variable en memoria que será la que manipulemos dentro del procedimiento. La Figura 177 muestra una representación gráfica de este proceso.

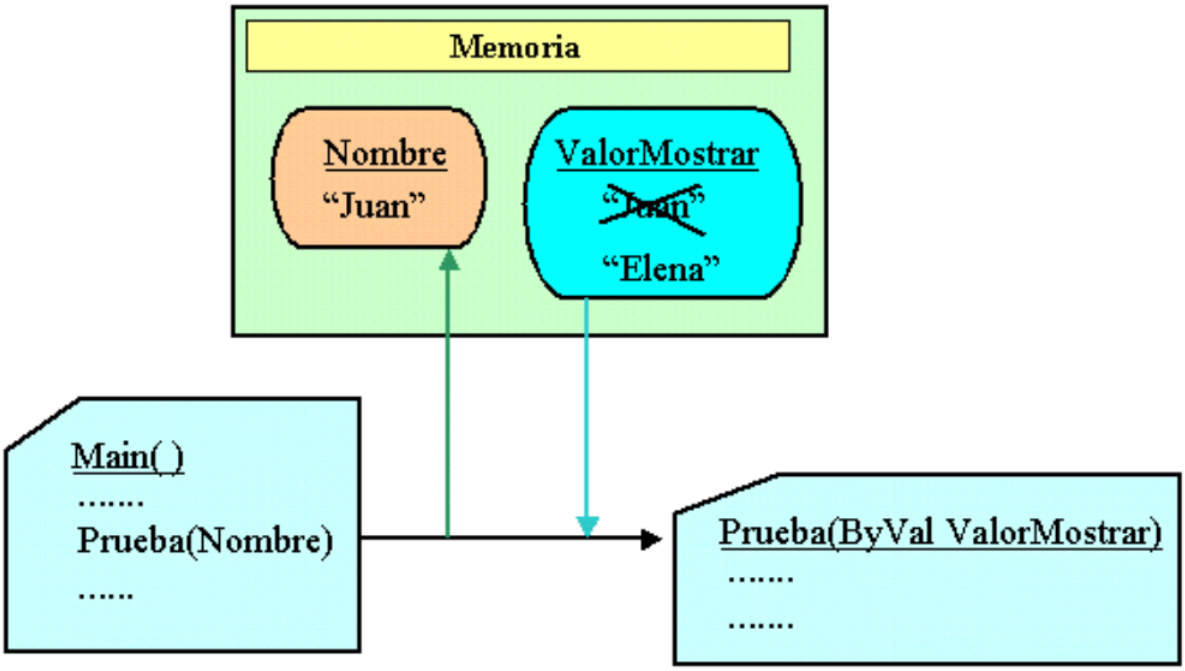

Figura 177. Esquema de gestión interna de variables en el paso por valor.

En el entorno de .NET Framework, por defecto, todos los parámetros son pasados por valor. Esto lo puede comprobar el lector de un modo muy simple: si al declarar un parámetro no especifica el tipo de paso, el IDE automáticamente situará junto a ese parámetro la palabra clave ByVal. Se recomienda siempre que sea posible el paso de parámetros por valor, ya que ayuda a generar un código más optimizado y contribuye a mejorar la encapsulación.

Paso por referencia (ByRef)

Cuando pasamos un parámetro por referencia a un procedimiento, si modificamos dentro del procedimiento la variable que contiene el parámetro, dichos cambios en este caso sí afectarán al código llamador. Debemos situar en este caso en el procedimiento, la palabra clave ByRef antes del nombre del parámetro. Cambiemos el código del anterior ejemplo, haciendo que en este caso, el parámetro sea pasado por referencia y observemos los resultados. Ver Código fuente 126.

```
Sub Main()
    Dim Nombre As String

    Nombre = "Juan"
    Console.WriteLine("Valor de la variable antes de llamar a Prueba(): {0}",
Nombre)

    ' llamamos a un procedimiento
    ' y le pasamos una variable por referencia
    Prueba(Nombre)

    ' el cambio realizado al parámetro en el procedimiento
    ' ha afectado a la variable Nombre, que aquí contiene
    ' el mismo valor que se asignó en el procedimiento
    Console.WriteLine("Valor de la variable al volver a Main(): {0}", Nombre)
    Console.ReadLine()
End Sub
```

```
Sub Prueba(ByRef ValorMostrar As String)
    ' modificamos el valor del parámetro
    ValorMostrar = "Elena"
    Console.WriteLine("Valor del parámetro dentro de Prueba(): {0}", ValorMostrar)
End Sub
```

Código fuente 126

Lo que ocurre en el fuente anterior a nivel de gestión interna en memoria de los parámetros es lo siguiente: cuando se realiza la llamada al procedimiento y se pasa el parámetro, el entorno detecta que se trata de un parámetro pasado por referencia, y tanto la variable del código llamador como la del procedimiento llamado utilizan la misma dirección de memoria o referencia hacia los datos, por lo que los cambios realizados en un procedimiento afectarán también al otro. La Figura 178 muestra una representación gráfica de este proceso.

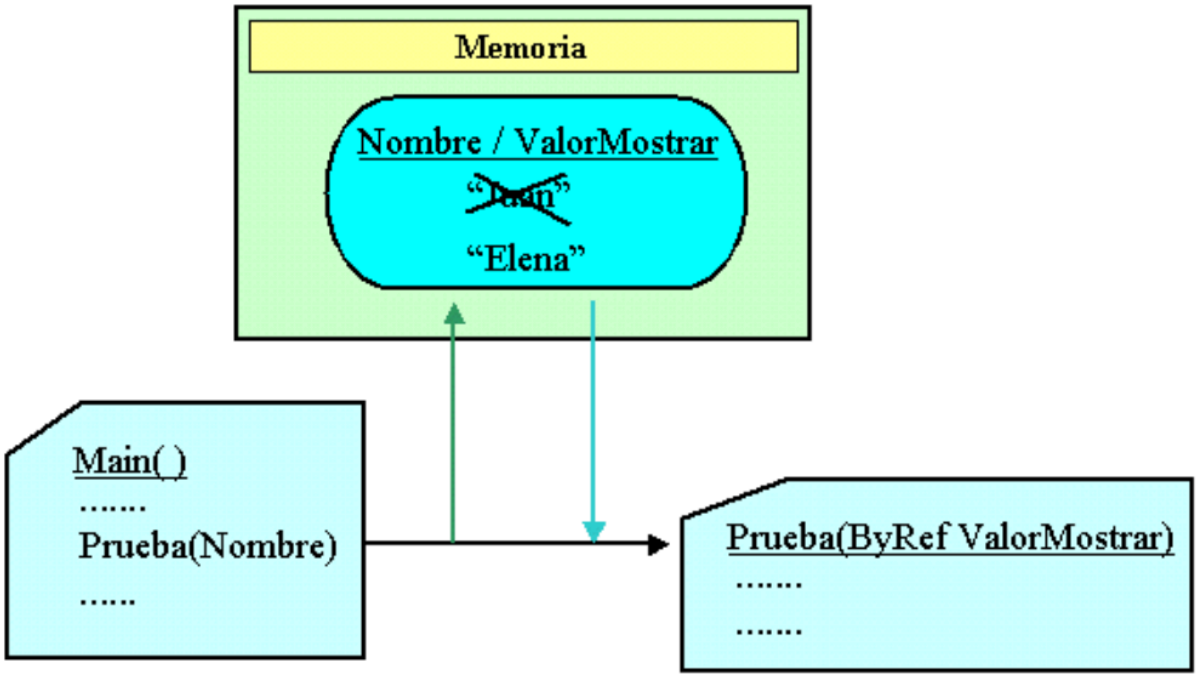

Figura 178. Esquema de gestión interna de variables en el paso por referencia.