

Entendiendo el paso de variables a métodos en C# con ref, params y out.

por Martín Márquez <xomalli@gmail.com>

En C# existen dos formas de pasar variables como parámetros a un método: por valor (Passing by value) que es la forma predeterminada y por referencia (Passing by reference, Passing by address). La definición del parámetro de un método consiste en el tipo de dato, el nombre del parámetro y si son más de uno se separan por coma como en los siguientes códigos:

```
int method(string p)
int method(string p, int q, bool r, float t);
```

Paso por valor (Passing by value)

Esta es la forma predeterminada en la que los métodos reciben los parámetros. Esto quiere decir que cuando un parámetro se recibe dentro del método, se crea una copia temporal de los datos, esta copia temporal actúa como una variable dentro del método con la que se realizan operaciones, sin afectar la variable original y una vez que el método termina su ejecución, la copia temporal es destruida, por lo tanto los cambios en el parámetro no afectan el valor de la variable original. Estos conceptos se observan mejor con el siguiente programa:

```
using System;

namespace Samples.PassingByValue
{
    class MainClass
    {
        public static int Main()
        {
            int a = 3;
            int b = 6;
            Console.WriteLine(Environment.NewLine);
            Console.WriteLine("Values before swapping.");
            Print(a, b);
            Console.WriteLine("Swapping...");
            Swap(a, b);
            Console.WriteLine("Values after swapping.");
            Print(a, b);
            Console.WriteLine(Environment.NewLine);
            return 0;
        }

        static void Print(int number1, int number2)
        {
            Console.WriteLine("(a) = {0}", number1);
            Console.WriteLine("(b) = {0}", number2);
        }
    }
}
```

```

}

static void Swap(int number1, int number2)
{
    int temp = number1;
    number1 = number2;
    number2 = temp;
}
}
}

```

Este programa intercambia el valor de unos enteros en donde se inicialmente se les asigna un valor, esos enteros después pasan como parámetros, se ejecuta el método y las variables conservan su valor inicial. Aquí muestro la siguiente salida:

Fig 1 El programa mostrando el paso por valor de variables

```

Ref : bash – Konsole
File Edit View Bookmarks Settings Help
martin@lori:~/Shared/Ref> mono Program.exe

Values before swapping.
(a) = 3
(b) = 6
Swapping...
Values after swapping.
(a) = 3
(b) = 6

martin@lori:~/Shared/Ref>

```



[Download el código fuente para Xamarin Studio o Visual Studio](#)

Paso por referencia (Passing by reference)

Hay 3 modificadores de parámetros en C# que sirven para pasar por referencia una variable a un método y estos son:

1. **ref**
2. **out**
3. **params**

Utilizando ref

Cuando se pasan las variables como parámetros por referencia, el parámetro es en realidad una referencia a la dirección de memoria de la variable que se recibe, por lo que los cambios hechos al parámetro afectan a la variable original. Para pasar un parámetro por referencia, se antepone a la definición del parámetro la palabra reservada **ref**. Para ilustrar este concepto utilizo el programa del ejemplo anterior que intercambia dos valores enteros, pero con la modificación de pasar las variables por referencia (con **ref**) al método no por valor. Aquí el código:

```
using System;

namespace Samples.PassingbyReference
{
    class MainClass
    {
        public static int Main()
        {
            int a = 3;
            int b = 6;
            Console.WriteLine(Environment.NewLine);
            Console.WriteLine("Values before swapping.");
            Print(a, b);
            Console.WriteLine("Swapping...");
            Swap(ref a, ref b);
            Console.WriteLine("Values after swapping.");
            Print(a, b);
            Console.WriteLine(Environment.NewLine);
            return 0;
        }

        static void Print(int number1, int number2)
        {
            Console.WriteLine("(a) = {0}", number1);
            Console.WriteLine("(b) = {0}", number2);
        }
    }
}
```

```
static void Swap(ref int number1, ref int number2)
{
    int temp = number1;
    number1 = number2;
    number2 = temp;
}
}
```

Al ejecutar este programa se produce el intercambio de variables que realiza el método *swap*, porque ya no trabaja con copias locales sino con las variables globales.

Fig 2 El programa mostrando el paso por valor de variables

```
Ref : bash - Konsole
File Edit View Bookmarks Settings Help
martin@lori:~/Shared/Ref> mono Program2.exe

Values before swapping.
(a) = 3
(b) = 6
Swapping...
Values after swapping.
(a) = 6
(b) = 3

martin@lori:~/Shared/Ref> 
```

Aquí el método *swap* con el modificador **ref** usado en la declaración de variables del método.

```
static void Swap(ref int number1, ref int number2)
{
    int temp = number1;
    number1 = number2;
    number2 = temp;
}
```

También se debe utilizar el modificador con sus argumentos al invocarlo.

```
Swap(ref a, ref b);
```



[Download el código fuente para Xamarin Studio o Visual Studio](#)

Utilizando params

En los ejemplos anteriores el número de parámetros está fijo en la declaración del método. Sin embargo, C# soporta el uso de un arreglo de parámetros (parameters array), la palabra reservada **params** permite pasar cualquier número variable de parámetros hacia un método sin necesidad de declararlos como un arreglo. Esta instrucción solo se requiere en la declaración del método. Con el siguiente programa muestro el uso de params para calcular el promedio de un arreglo de números y para armar una cadena.

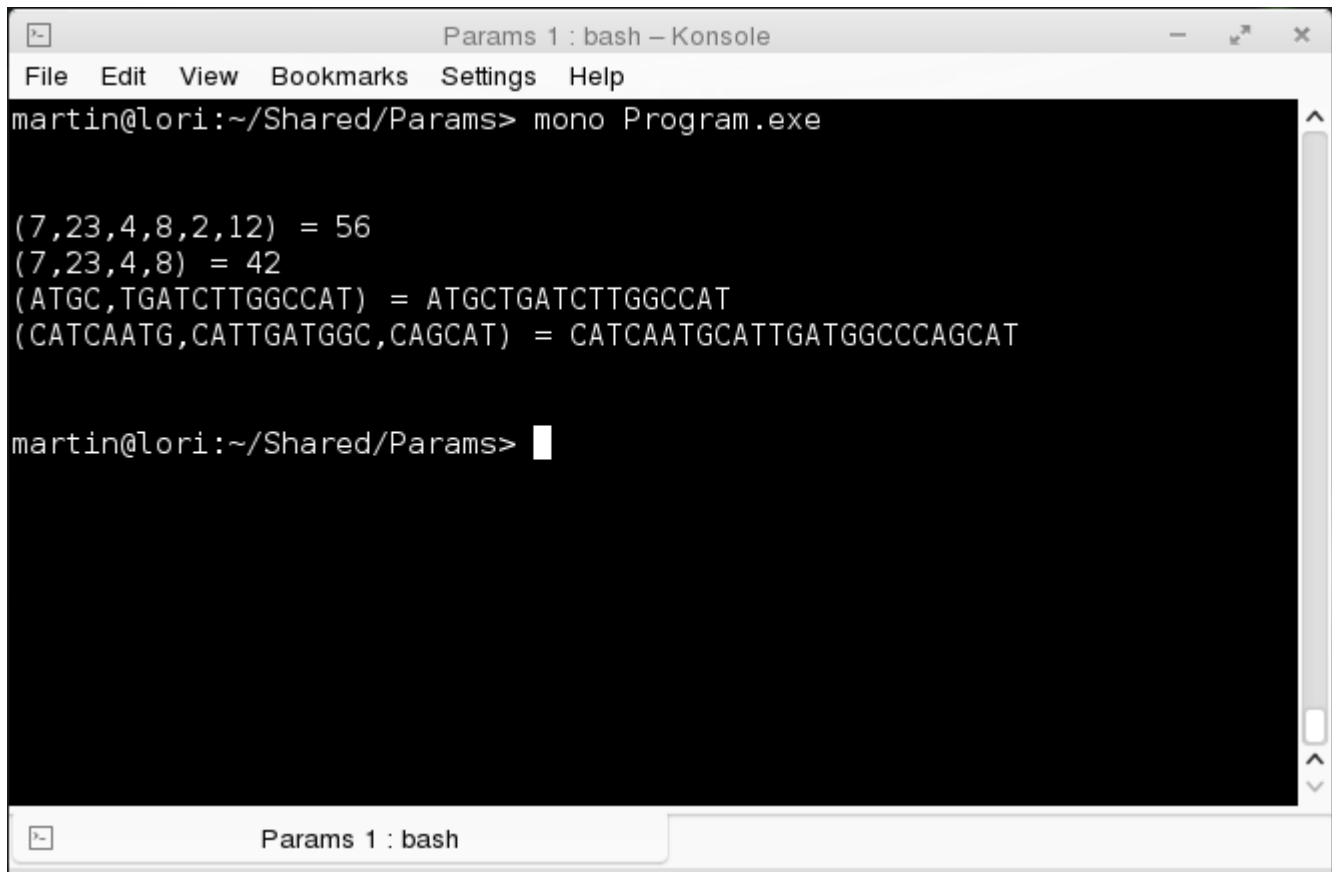
```
using System;

namespace Samples.Params
{
    class MainClass
    {
        public static int Main(string[] args)
        {
            Console.WriteLine(Environment.NewLine);
            Console.WriteLine("Sum(7,23,4,8,2,12) = {0}", Sum(7, 23, 4, 8, 2, 12));
            Console.WriteLine("Sum(7,23,4,8) = {0}", Sum(7, 23, 4, 8));
            Console.WriteLine("ATGC,TGATCTTGCCAT = ({0})", Assembly("ATGC",
"TGATCTTGCCAT"));
            Console.WriteLine("CATCAATG,CATTGATGGC,CAGCAT = ({0})", Assembly("CATCAATG",
"CATTGATGGC", "CAGCAT"));
            Console.WriteLine(Environment.NewLine);
            return 0;
        }

        static float Sum(params int[] data)
        {
            int sum = 0;
            for (int i = 0; i < data.Length; i++)
                sum += data[i];
            return sum;
        }

        static string Assembly(params string[] sequences)
        {
            string s = null;
            for (var i = 0; i < sequences.Length; i++)
                s += sequences[i];
            return s;
        }
    }
}
```

Fig 3 El programa mostrando el uso de *params* en un método.



```
Params 1 : bash - Konsole
File Edit View Bookmarks Settings Help
martin@lori:~/Shared/Params> mono Program.exe

(7,23,4,8,2,12) = 56
(7,23,4,8) = 42
(ATGC,TGATCTTGGCCAT) = ATGCTGATCTTGGCCAT
(CATCAATG,CATTGATGGC,CAGCAT) = CATCAATGCATTGATGGCCCAGCAT

martin@lori:~/Shared/Params> 
```



[Download el código fuente para Xamarin Studio o Visual Studio](#)

Utilizando output parameters

Un método de forma predeterminada solo puede devolver un valor mediante la instrucción **return**, al menos que se utilicen los output parameters (parámetros de salida), con esta característica pueden devolver tantos valores de salida como parámetros de salida tengan, no es necesario asignar valores a esos parámetros afuera del método sino se debe asignar el valor dentro del método que los va a utilizar o de lo contrario el compilador marcará un error.

Para utilizar los output parameters se debe anteponer la palabra reservada: **out** en la declaración de cada parámetro. Como se muestra en el siguiente código:

```
static void SetEnviromentProperties(out string currentDirectory,
    out string machineName,
    out string osVersion,
    out string userName,
    out int exitCode)
```

Bien con el siguiente programa mostraré el uso de esta característica.

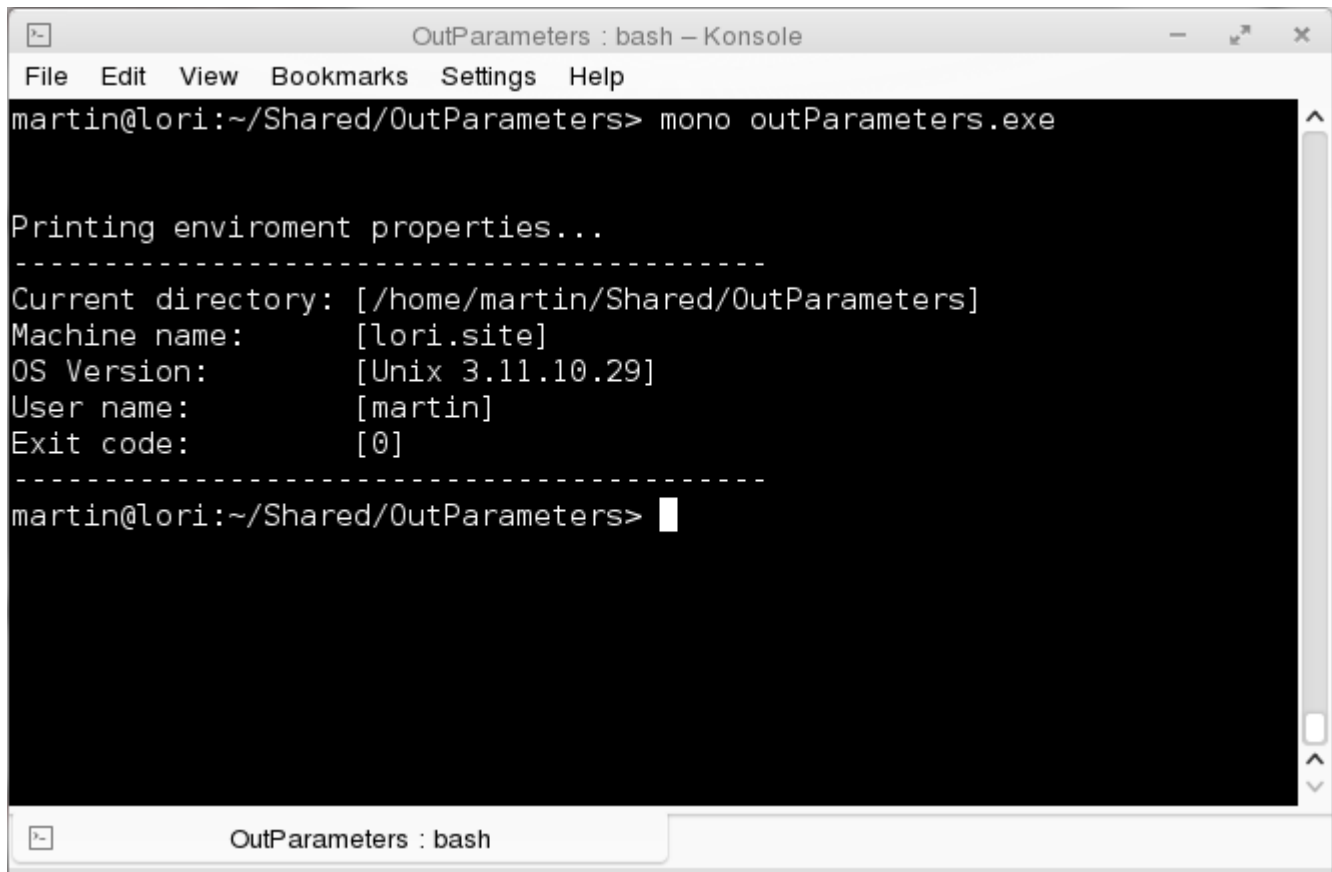
```
using System;

namespace Samples.OutputParameters
{
    class Program
    {
        public static int Main(string[] args)
        {
            string currentDirectory;
            string machineName;
            string osVersion;
            string userName;
            int exitCode;
            Console.WriteLine(Environment.NewLine);
            SetEnviromentProperties(out currentDirectory,
                out machineName,
                out osVersion,
                out userName,
                out exitCode);
            Console.WriteLine("Printing enviroment properties...");
            Console.WriteLine("-----");
            Console.WriteLine("Current directory: [{0}]", currentDirectory);
            Console.WriteLine("Machine name:      [{0}]", machineName);
            Console.WriteLine("OS Version:       [{0}]", osVersion);
            Console.WriteLine("User name:        [{0}]", userName);
            Console.WriteLine("Exit code:        [{0}]", exitCode);
            Console.WriteLine("-----");
            return 0;
        }

        static void SetEnviromentProperties(out string currentDirectory,
            out string machineName,
            out string osVersion,
            out string userName,
            out int exitCode)
        {
            //setting values to the variables
            currentDirectory = Environment.CurrentDirectory;
            machineName = Environment.MachineName;
            osVersion = Environment.OSVersion.ToString();
            userName = Environment.UserName;
            exitCode = Environment.ExitCode;
        }
    }
}
```

Al ejecutar el programa, se vera la siguiente salida:

Fig 4 El programa mostrando la utilización de *output parameters*



```
OutParameters : bash - Konsole
File Edit View Bookmarks Settings Help
martin@lori:~/Shared/OutParameters> mono outParameters.exe

Printing enviroment properties...
-----
Current directory: [/home/martin/Shared/OutParameters]
Machine name:      [lori.site]
OS Version:       [Unix 3.11.10.29]
User name:        [martin]
Exit code:        [0]
-----
martin@lori:~/Shared/OutParameters> 
```

Al ejecutarse el método *SetEnviromentProperties* sus correspondientes argumentos también deben tener la palabra reservada **out**, para que dentro del método se les asigne el valor correspondiente.

```
static void SetEnviromentProperties(out string currentDirectory,
    out string machineName,
    out string osVersion,
    out string userName,
    out int exitCode)
```

Finalmente hay que recordar que un método que define output parameters DEBE asignar un valor a los parámetros **out** antes de que el método termine de ejecutarse, de lo contrario el compilador marcará un error.



[Download el código fuente para Xamarin Studio o Visual Studio](#)

Este documento está protegido bajo la licencia de documentación libre *Free Documentacion License* del Proyecto GNU, para consulta ver el sitio <http://www.gnu.org/licenses/fdl.txt> , toda persona que lo desee está autorizada a usar, copiar y modificar este documento según los puntos establecidos en la «Licencia FDL»