



Fundamentos de Programação

Wesley Dias Maciel

Passagem de Parâmetros

- **Passagem de Parâmetros do Tipo Valor.**
 - Exemplo: char, string, bool, int, long, float, double, decimal.
 - Mecanismo: cópia.
- Passagem de Parâmetro por Valor.
- Passagem de Parâmetro por Referência.
 - Modificadores **ref**, **in** e **out**.
- **Passagem de Parâmetros do Tipo Referência.**
 - Exemplo: objetos, como vetores e matrizes.
 - Mecanismo: apelido.
- **params.**

Passagem de Parâmetros



Passagem de Parâmetros



Passagem de Parâmetros

Parâmetros



Passagem de Parâmetros



Parâmetros

- Parâmetros:
 - Aumentam poder das abstrações.
- Exemplo:

```
//Definição do método:  
static double area (double raio)  
{  
    return Math.PI * raio * raio;  
}  
  
area (5);      //Chamada do método.  
area (a + b);  //Chamada do método.
```

Parâmetros

- Definições:

- **Argumento:**

- **Valor** que é passado para o método.
 - Exemplo: **5** e o **resultado** da avaliação da expressão ***a + b***.

- **Parâmetro de chamada** ou **parâmetro real:**

- **Expressão** que produz o argumento.
 - Exemplo: **5** e ***a + b***.

- **Parâmetro formal:**

- **Identificador** que denota o argumento no interior da função (na definição da função).
 - Exemplo: ***raio*** .

Exemplo:

```
//Definição do método:  
static double area (double raio)  
{  
    return Math.PI * raio * raio;  
}  
area (5);      //Chamada do método.  
area (a + b);  //Chamada do método.
```


Passagem de Parâmetros do Tipo Valor

Passagem de Parâmetro por Valor

Passagem de Parâmetro por Valor

```
using System;
```

```
class Programa
```

 $\{$

```
static void Main ()
```

{

```
int num = 5;
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

```
dobrar (num);
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

}

```
static void dobrar (int valor)
```

 $\{$

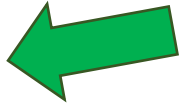
```
valor *= 2;
```

}

}

OBS:

- Criação da variável "**num**" que é do tipo "int".



num

[illegible]

Passagem de Parâmetro por Valor

```
using System;
```

```
class Programa
```

 $\{$

```
static void Main ()
```

{

```
int num = 5;
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

dobrar (num); 

```
Console.WriteLine ("\n\t num = {0}", num);
```

}

```
static void dobrar (int valor)
```

{

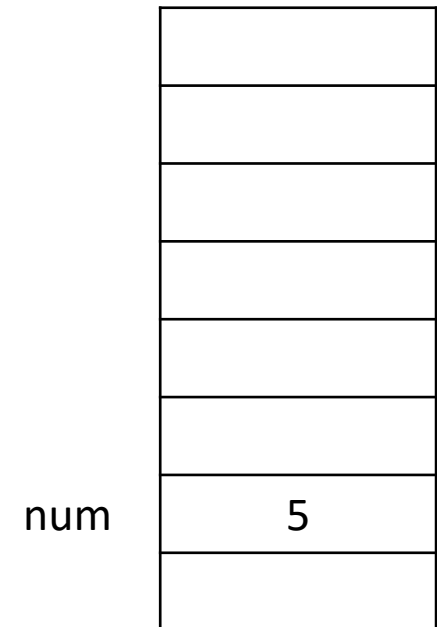
```
valor *= 2;
```

}

}

OBS:

- Chamada do método "**dobrar ()**" passando a variável "**num**" como parâmetro: passagem por valor.



Passagem de Parâmetro por Valor

using System;

```
class Programa
```

```
{
```

```
    static void Main ()
```

```
    {
```

```
        int num = 5;
```

```
        Console.WriteLine ("\n\t num = {0}", num);
```

```
        dobrar (num);
```

```
        Console.WriteLine ("\n\t num = {0}", num);
```

```
    }
```

```
    static void dobrar (int valor)
```

```
    {
```

```
        valor *= 2;
```

```
    }
```

```
}
```

OBS: mecanismo de cópia:

- As variáveis "**num**" e "**valor**" estão em posições diferentes de memória.
- O conteúdo da variável "**num**" é copiado para a variável "**valor**".
- Duas posições diferentes de memória são usadas no processamento.



Passagem de Parâmetro por Valor

using System;

```
class Programa
```

```
{
```

```
    static void Main ()
```

```
    {
```

```
        int num = 5;
```

```
        Console.WriteLine ("\n\t num = {0}", num);
```

```
        dobrar (num);
```

```
        Console.WriteLine ("\n\t num = {0}", num);
```

```
    }
```

```
    static void dobrar (int valor)
```

```
    {
```

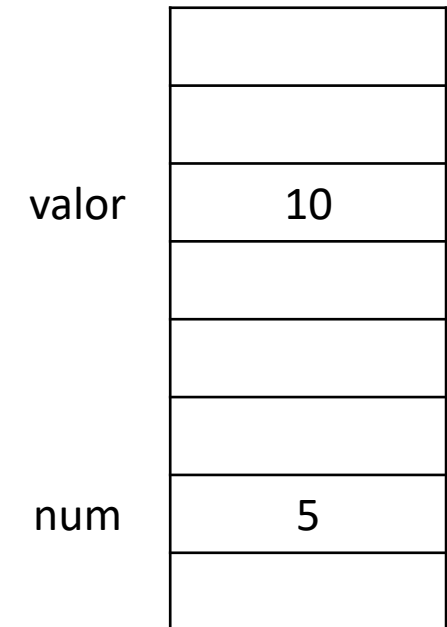
```
        valor *= 2;
```

```
    }
```

```
}
```

OBS:

- O conteúdo da variável "**valor**" é modificado.
- O conteúdo da variável "**num**" continua inalterado.
 - Processamento realizado em posições diferentes de memória.



Passagem de Parâmetro por Valor

using System;

class Programa

{

static void Main ()

{

int num = 5;

Console.WriteLine ("\n\t num = {0}", num);

dobrar (num);

Console.WriteLine ("\n\t num = {0}", num);

}

static void dobrar (int valor)

{

valor *= 2;

}

}

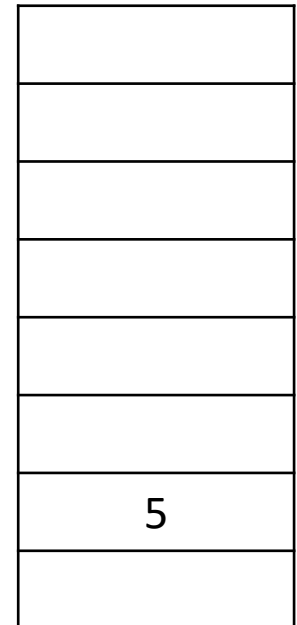
OBS:

- A variável "**valor**" é destruída, eliminada da memória principal.
- Uma posição de memória é liberada.

num = 5

num = 5

num



Passagem de Parâmetro por Valor

```
using System;

class Programa
{
    static void Main ()
    {
        string nome = "Marcos";
        Console.WriteLine ("\n\t nome = {0}", nome);
        modificar (nome);
        Console.WriteLine ("\n\t nome = {0}", nome);
    }
}
```

```
static void modificar (string str)
{
    Console.WriteLine ("\n\t str = {0}", str);
    str += ".Modificado";
    Console.WriteLine ("\n\t str = {0}", str);
}
}
```

```
nome = Marcos
str = Marcos
str = Marcos.Modificado
nome = Marcos
```



Passagem de Parâmetro por Referência

Modificador de Parâmetro **ref**

- A palavra-chave **ref** faz com que os argumentos sejam passados por referência, não por valor.
- Ela torna o parâmetro formal um apelido para o argumento.
 - O argumento tem que ser uma variável.
- Em resumo:
 - Qualquer alteração no parâmetro é feita também no argumento.
 - Os argumentos **ref** precisam ser iniciados antes de serem passados como parâmetro.
 - Esses parâmetros são as entradas e os retornos, saídas, do método.
 - Os argumentos **ref** podem ser modificados pelo método chamado.
 - Permitem edição.
 - São do tipo "leitura e escrita".

Modificador de Parâmetro **ref**

- Microsoft: <https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/ref>

 Filtrar por título

parametro)

ref

out (modificador de parâmetro)

> Palavras-chave de namespace

> Palavras-chave de teste de tipo

> Tipo genérico de palavras-chave de restrição


> Palavras-chave de acesso

> Palavras-chave literais

> Palavras-chave contextuais

> Palavras-chave de consulta

> Operadores e expressões C#

 Baixar PDF

Passando um argumento por referência

Quando usado na lista de parâmetros do método, a palavra-chave `ref` indica que um argumento é passado por referência, não por valor. A palavra-chave `ref` torna o parâmetro formal um alias para o argumento, que deve ser uma variável. Em outras palavras, qualquer operação no parâmetro é feita no argumento. Por exemplo, se o chamador passar uma expressão de variável local ou uma expressão de acesso a elemento de matriz, e o método chamado substituir o objeto ao qual o parâmetro de `ref` se refere, então a variável local do chamador ou o elemento de matriz agora se refere ao novo objeto quando o método retorna.

⚠ Observação

Não confunda o conceito de passar por referência com o conceito de tipos de referência. Os dois conceitos não são iguais. Um parâmetro de método pode ser modificado por `ref`, independentemente de ele ser um tipo de valor ou um tipo de referência. Não há nenhuma conversão boxing de um tipo de valor quando ele é passado por referência.

Para usar um parâmetro `ref`, a definição do método e o método de chamada devem usar

Esta página é útil?

 Yes  No

Neste artigo

[Passando um argumento por referência](#)

[Passando um argumento por referência: um exemplo](#)

[Valores retornados por referência](#)

[Ref locais](#)

[Locais somente leitura de referência](#)

[Um exemplo de ref returns e ref locals](#)

Modificador de Parâmetro **ref**

```
using System;
```

OBS: mecanismo de referência:

- **"valor"** é um apelido para **"num"**.

```
class Programa
```

$$\{$$

```
static void Main ()
```

{

```
int num = 5;
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

```
dobrar (ref num);
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

}

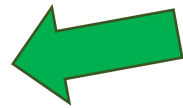
```
static void dobrar (ref int valor)
```

{

```
valor *= 2;
```

}

}



valor, num

[illegible]

Modificador de Parâmetro **ref**

```
using System;
```

```
class Programa
```

 $\{$

```
static void Main ()
```

{

```
int num = 5;
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

```
dobrar (ref num);
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

}

```
static void dobrar (ref int valor)
```

{

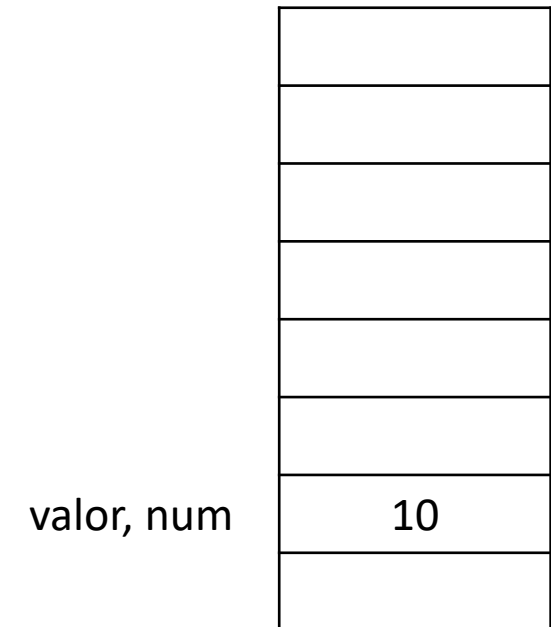
```
valor *= 2;
```

}

}

OBS: mecanismo de referência:

- "**valor**" é um apelido para "**num**".
- O processamento é realizado na mesma posição de memória.
 - Apenas uma posição de memória é usada.



Modificador de Parâmetro **ref**

```
using System;
```

OBS:

- O apelido "**valor**" é destruído.
- A posição de memória é preservada com "**num**".

```
class Programa
```

 $\{$

```
static void Main ()
```

$$\{$$

```
int num = 5;
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

```
dobrar (ref num);
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

}

```
static void dobrar (ref int valor)
```

 $\{$

```
valor *= 2;
```

}

}

```
num = 5
num = 10
```

num

[illegible]

Modificador de Parâmetro **in**

- A palavra-chave **in** faz com que os argumentos sejam passados por referência.
- Ela torna o parâmetro formal um apelido para o argumento.
 - O argumento tem que ser uma variável.
- Entretanto:
 - Os argumentos **in** precisam ser iniciados antes de serem passados como parâmetro, pois seus valores serão lidos no método chamado.
 - Esses parâmetros são as entradas do método.
 - Os argumentos **in** não podem ser modificados pelo método chamado.
 - Não permitem edição.
 - São do tipo "somente leitura".

Modificador de Parâmetro **in**

- Microsoft: <https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/in-parameter-modifier>

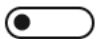
Docs / .NET / Guia do C# / Referência de linguagem / Palavras-chave

Indicador

Comentários

Compartilhar

Ler em inglês



ⓘ Algumas partes deste tópico podem ter sido traduzidas automaticamente.



Filtrar por título

params

in (modificador de parâmetro)

ref

out (modificador de parâmetro)

> Palavras-chave de namespace

> Palavras-chave de teste de tipo

> Tipo genérico de palavras-

Baixar PDF

Modificador de parâmetro in (referência do C#)

19/03/2020 • 5 minutos para ler • 🌐 🚩

A palavra-chave **in** faz com que os argumentos sejam passados por referência. Ela torna o parâmetro formal um alias para o argumento, que deve ser uma variável. Em outras palavras, qualquer operação no parâmetro é feita no argumento. E como as palavras-chave **ref** ou **out**, exceto que os argumentos **in** não podem ser modificados pelo método chamado. Enquanto os argumentos **ref** podem ser modificados, os argumentos **out** devem ser modificados pelo método chamado, e essas modificações podem ser observadas no contexto da chamada.

C#

Copiar

Executar

```
int readonlyArgument = 44;  
InArgExample(readonlyArgument);
```

Esta página é útil?

👍 Yes 🗨 No

Neste artigo

Regras de resolução de sobrecarga

Limitações em parâmetros in

Especificação da Linguagem C#

Confira também

Modificador de Parâmetro **in**

```
using System;
```

OBS:

- **ERRO DE COMPILAÇÃO:**
 - O método tenta alterar o conteúdo de "**valor**".

```
class Programa
```

```
{
```

```
    static void Main ()
```

```
    {
```

```
        int num = 5;
```

```
        Console.WriteLine ("\n\t num = {0}", num);
```

```
        dobrar (in num);
```

```
        Console.WriteLine ("\n\t num = {0}", num);
```

```
    }
```

```
    static void dobrar (in int valor)
```

```
    {
```

```
        valor *= 2;
```

```
    }
```

```
}
```

```
using System;

0 referências
class Programa
{
    0 referências
    static void Main ()
    {
        int num = 5;
        Console.WriteLine ("\n\t num = {0}", num);
        dobrar (in num);
        Console.WriteLine ("\n\t num = {0}", num);
    }
    1 referência
    static void dobrar (in int valor)
    {
        valor *= 2;
    }
}
```

[🔍] (parâmetro) **in** int valor

Não é possível atribuir a variável 'in int' porque não é uma variável somente leitura

Modificador de Parâmetro **in**

using System;

OBS:

- **OK!**
 - A variável "**valor**" é um apelido para a variável "**num**".
 - A variável "**resp**" é um apelido para a variável "**retorno**".
 - A variável "**valor**" não é editável.
 - A variável "**resp**" é editável.

```
class Programa
{
    static void Main ()
    {
        int num = 5, retorno = 0;
        Console.WriteLine ("\n\t num = {0} retorno = {1}", num, retorno);
        dobrar (in num, ref retorno);
        Console.WriteLine ("\n\t num = {0} retorno = {1}", num, retorno);
    }
    static void dobrar (in int valor, ref int resp)
    {
        resp = valor;
        resp *= 2;
    }
}
```

```
num = 5  retorno = 0
num = 5  retorno = 10
```

Editável!

↑
resp, retorno

↓
valor, num

Não editável!

10
5

Modificador de Parâmetro **in**

using System;

OBS:

- **OK!**

- As variáveis "**valor**" e "**resp**" são apelidos para a variável "**num**".
 - Então, a variável "**num**" tem dois apelidos: "**valor**" e "**resp**".
- A variável "**valor**" não é editável.
- A variável "**resp**" é editável.

```
class Programa
```

 $\{$

```
static void Main ()
```

$$\{$$

```
int num = 5;
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

```
dobrar (in num, ref num);
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

}

```
static void dobrar (in int valor, ref int resp)
```

 $\{$

```
resp = valor;
```

```
resp *= 2;
```

}

}

```
num = 5
num = 10
```

Editável!

resp, valor, num

Não editável!

[illegible]

Modificador de Parâmetro **in**

```
using System;

class Programa
{
    static void Main ()
    {
        int num = 5;
        Console.WriteLine ("\n\t num: {0}", num);
        dobrar (in num, ref num);
        Console.WriteLine ("\n\t num: {0}", num);
    }
}
```

```
static void dobrar (in int valor, ref int resp)
{
    Console.WriteLine ("\n\t valor: {0}", valor);
    Console.WriteLine ("\n\t resp: {0}", resp);
    resp = valor;
    resp *= 2;
    Console.WriteLine ("\n\t valor: {0}", valor);
    Console.WriteLine ("\n\t resp: {0}", resp);
}
}
```

```
num: 5
valor: 5
resp: 5
valor: 10
resp: 10
num: 10
```

Modificador de Parâmetro **out**

- A palavra-chave **out** faz com que os argumentos sejam passados por referência.
- Ela torna o parâmetro formal um apelido para o argumento.
 - O argumento tem que ser uma variável.
- Entretanto:
 - Os argumentos **out** não precisam ser iniciados, pois seus valores não serão lidos no método chamado.
 - Entretanto, o método chamado tem que atribuir um valor a esses parâmetros antes que o método seja finalizado.
 - Esses parâmetros são os retornos, saídas, do método.
 - Os argumentos **out** são modificados pelo método chamado.
 - Permitem edição.
 - São do tipo "somente escrita".

Modificador de Parâmetro out

- Microsoft: <https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/out-parameter-modifier>

[Docs](#) / [.NET](#) / [Guia do C#](#) / [Referência de linguagem](#) / [Palavras-chave](#)

[Indicador](#) [Comentários](#) [Compartilhar](#) [Ler em inglês](#)

ⓘ Algumas partes deste tópico podem ter sido traduzidas automaticamente.

Filtrar por título

ref

out (modificador de parâmetro)

- > Palavras-chave de namespace
- > Palavras-chave de teste de tipo
- > Tipo genérico de palavras-chave de restrição
- > Palavras-chave de acesso
- > Palavras-chave literais

Baixar PDF

Modificador de parâmetro out (Referência de C#)

19/03/2020 • 3 minutos para ler • 🌐 🔄 🏠

A palavra-chave `out` faz com que os argumentos sejam passados por referência. Ela torna o parâmetro formal um alias para o argumento, que deve ser uma variável. Em outras palavras, qualquer operação no parâmetro é feita no argumento. É como a palavra-chave `ref`, exceto pelo fato de que `ref` requer que a variável seja inicializada antes de ser passada. Também é como a palavra-chave `in`, exceto que `in` não permite que o método chamado modifique o valor do argumento. Para usar um parâmetro `out`, a definição do método e o método de chamada devem usar explicitamente a palavra-chave `out`. Por exemplo:

C#

Copiar

Executar

Esta página é útil?

[Yes](#) [No](#)

Neste artigo

[Declarando parâmetros out](#)

[Chamando um método com um argumento out](#)

[Especificação da Linguagem C#](#)

[Confira também](#)

Modificador de Parâmetro **out**

```
using System;
```

OBS:

- **ERRO DE COMPILAÇÃO:**
 - O método finaliza sem atribuir um conteúdo para "**valor**".

```
class Programa
```

```
{
```

```
    static void Main ()
```

```
    {
```

```
        int num;
```

```
        dobrar (out num);
```

```
        Console.WriteLine ("\n\t num = {0}", num);
```

```
    }
```

```
    static void dobrar (out int valor)
```

```
    {
```

```
        Console.WriteLine ("\n\t valor = {0}", valor);
```

```
    }
```

```
}
```

```
using System;
```

0 referências

```
class Programa
```

```
{
```

0 referências

```
    static void Main ()
```

```
    {
```

```
        int num;
```

```
        dobrar (out num);
```

```
        Console.WriteLine ("\n\t num = {0}", num);
```

```
    }
```

1 referência

```
    static void dobrar (out int valor)
```

```
    {
```

```
        Console.WriteLine ("\n\t valor = {0}", valor);
```

```
    }
```

```
}
```

(parâmetro) out int valor

Uso do parâmetro out não atribuído "valor"

Modificador de Parâmetro **out**

```
using System;
```

OBS:

- **ERRO DE COMPILAÇÃO:**

- A variável "**num**" foi iniciada, mas o método finaliza sem atribuir um conteúdo para "**valor**".

```
class Programa
```

```
{
```

```
    static void Main ()
```

```
    {
```

```
        int num = 5;
```

```
        dobrar (out num);
```

```
        Console.WriteLine ("\n\t num = {0}", num);
```

```
    }
```

```
    static void dobrar (out int valor)
```

```
    {
```

```
        Console.WriteLine ("\n\t valor = {0}", valor);
```

```
    }
```

```
}
```

```
using System;

0 referências
class Programa
{
    0 referências
    static void Main ()
    {
        int num = 5;
        dobrar (out num);
        Console.WriteLine ("\n\t num = {0}", num);
    }
    1 referência
    static void dobrar (out int valor)
    {
        Console.WriteLine ("\n\t valor = {0}", valor);
    }
}
```

[?] (parâmetro) `out int valor`

Uso do parâmetro out não atribuído "valor"

Modificador de Parâmetro **out**

```
using System;
```

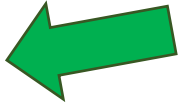
```
class Programa
```

```
{
```

```
    static void Main ()
```

```
    {
```

```
        int num;
```



```
        dobrar (out num);
```

```
        Console.WriteLine ("\n\t num = {0}", num);
```

```
    }
```

```
    static void dobrar (out int valor)
```

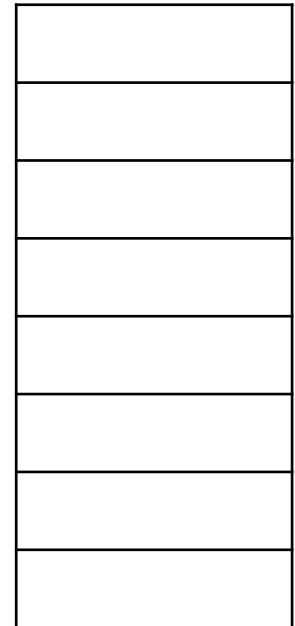
```
    {
```

```
        valor = 10;
```

```
    }
```

```
}
```

num



Modificador de Parâmetro out

```
using System;
```

```
class Programa
```

 $\{$

```
static void Main ()
```

 $\{$

```
int num;
```

```
dobrar (out num);
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

}

```
static void dobrar (out int valor)
```

$$\{$$

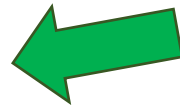
```
valor = 10;
```

}

}

OBS: mecanismo de referência:

- **"valor"** é um apelido para **"num"**.



valor, num

[illegible]

Modificador de Parâmetro out

```
using System;
```

```
class Programa
```

 $\{$

```
static void Main ()
```

{

```
int num;
```

```
dobrar (out num);
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

}

```
static void dobrar (out int valor)
```

{

```
valor = 10;
```

}

}

OBS: mecanismo de referência:

- "**valor**" é um apelido para "**num**".
- O processamento é realizado na mesma posição de memória.
 - Apenas uma posição de memória é usada.

valor, num

[illegible]

Modificador de Parâmetro out

```
using System;
```

```
class Programa
```

$$\{$$

```
static void Main ()
```

{

```
int num;
```

```
dobrar (out num);
```

```
Console.WriteLine ("\n\t num = {0}", num);
```

}

```
static void dobrar (out int valor)
```

{

```
valor = 10;
```

}

}

OBS: mecanismo de referência:

- O apelido "**valor**" é destruído, eliminado da memória.

```
num = 10
```

10

valor, num

Passagem de Parâmetros do Tipo Referência

Passagem de Parâmetros do Tipo Referência - Vetor

```
using System;

class Programa
{
    static void Main ()
    {
        int[] vetor = { 1, 2, 3, 4, 5 };

        imprimir (vetor);
        modificar (vetor);
        imprimir (vetor);
    }
}

static void imprimir (int[] vet)
{
    Console.Write ("\n\t");
    for (int i = 0; i < vet.Length; i++)
        Console.Write ("{0}  ", vet[i]);
    Console.WriteLine ();
}

static void modificar (int[] vet)
{
    for (int i = 0; i < vet.Length; i++)
        vet[i] = 9;
}
```

1	2	3	4	5
9	9	9	9	9

Passagem de Parâmetros do Tipo Referência - Matriz

```
using System;
```

```
class Programa
```

```
{
```

```
    static void Main ()
```

```
    {
```

```
        int[,] matriz = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
```

```
        imprimir (matriz);
```

```
        modificar (matriz);
```

```
        imprimir (matriz);
```

```
    }
```

```
static void imprimir (int[,] mat)
```

```
{
```

```
    Console.Write ("\n\t");
```

```
    for (int i = 0; i < 3; i++)
```

```
    {
```

```
        for (int j = 0; j < 3; j++)
```

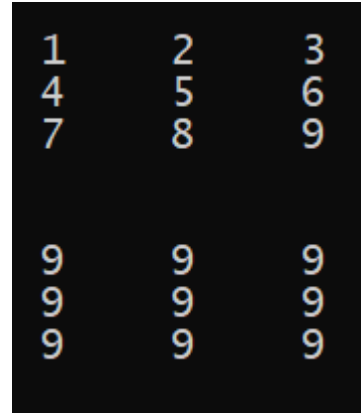
```
            Console.Write (" {0} ", mat[i, j]);
```

```
            Console.Write ("\n\t");
```

```
        }
```

```
    Console.WriteLine ();
```

```
}
```



1	2	3
4	5	6
7	8	9
9	9	9
9	9	9
9	9	9

```
static void modificar (int[,] mat)
```

```
{
```

```
    for (int i = 0; i < 3; i++)
```

```
        for (int j = 0; j < 3; j++)
```

```
            mat[i, j] = 9;
```

```
}
```

```
}
```


Passagem de Parâmetros do Tipo Referência - Objeto

```
using System;

class Programa
{
    static void Main ()
    {
        Aluno a = new Aluno ();

        Console.WriteLine ("\n\t nome = {0}", a.nome);
        modificar (a);
        Console.WriteLine ("\n\t nome = {0}", a.nome);
    }
}
```

```
static void modificar (Aluno x)
{
    x.nome = "Caio";
}

public class Aluno {
    public string nome = "Ana";
}
```

```
nome = Ana
nome = Caio
```

params

params

- Permite que um método receba um número indeterminado de parâmetros:
 - Zero ou mais parâmetros.
- Não permite o uso dos modificadores **ref**, **in** e **out**.
- Caso o método possua outros parâmetros além do parâmetro "**params**", o parâmetro "**params**" deve ser o último da lista de parâmetros.

params

```
using System;
```

```
class Programa
```

```
{
```

```
    static void Main ()
```

```
    {
```

```
        int parametro0 = 10, parametro1 = 20, parametro2 = 30;
```

```
        Console.WriteLine ("\n\t Primeira chamada: "); metodo ();
```

```
        Console.WriteLine ("\n\t Segunda chamada: "); metodo (parametro0);
```

```
        Console.WriteLine ("\n\t Terceira chamada: "); metodo (parametro0, parametro1);
```

```
        Console.WriteLine ("\n\t Quarta chamada: "); metodo (parametro0, parametro1, parametro2);
```

```
    }
```

```
static void metodo (params int[] parametros)
```

```
{
```

```
    if (parametros.Length > 0)
```

```
        for (int i = 0; i < parametros.Length; i++)
```

```
            Console.WriteLine ("\t parametros[{0}] = {1}", i, parametros[i]);
```

```
    Console.WriteLine ();
```

```
}
```

```
}
```

```
Primeira chamada:
```

```
Segunda chamada:      parametros[0] = 10
```

```
Terceira chamada:     parametros[0] = 10      parametros[1] = 20
```

```
Quarta chamada:       parametros[0] = 10      parametros[1] = 20      parametros[2] = 30
```

Exercício 01:

Escreva um algoritmo em C# que:

- 1) Receba um número a partir do teclado.
- 2) Possua um método que receba o número como argumento e retorne o dobro desse número.
- 3) O método deve implementar a passagem de parâmetro por referência, "leitura e escrita", para realizar o retorno do resultado.

Exercício 02:

Escreva um algoritmo em C# que:

- 1) Receba dois números a partir do teclado.
- 2) Possua um método que receba os dois números como argumento e retorne o quociente e o resto da divisão do primeiro pelo segundo número.
- 3) Os dois números devem ser informados para o método como "somente leitura".
- 4) O quociente e o resto devem ser retornados pelo método como "somente escrita", para realizar o retorno dos resultados.

Exercício 03:

Escreva um algoritmo em C# que:

- 1) Possua um método que receba uma quantidade indeterminada de números como argumento e retorne a média aritmética dos números informados.
- 2) A média deve ser retornada pelo método como "somente escrita", para realizar o retorno do resultado.