

SISTEMAS DISTRIBUÍDOS

C# - Sintaxe Básica
Visual Studio

PROFESSOR HILDEBERTO MELO



Apresentação

- Técnico em Desenvolvimento de Sistemas - Ibratec, Recife-PE
- Bacharel em Sistemas de Informação – FIR, Recife-PE
- Especialista em Docência no Ensino Superior – Faculdade Maurício de Nassau, Recife-PE
- Mestre em Ciência da Computação – UFPE/CIN, Recife-PE
- *Currículo Lattes* <http://lattes.cnpq.br/0759508594425296>)
- *Homepage* <https://sites.google.com/site/hildebertomelo/>



Disciplinas Lecionadas

- Desenvolvimento de Aplicações Desktop
- Programação Orientada a Objetos
- Estrutura de Dados
- Tecnologia da Informação & Sociedade
- Sistemas Operacionais
- Sistemas Distribuídos
- Introdução a Informática
- Lógica de Programação
- Informática Aplicada a Saúde
- Banco de Dados
- Projeto de Banco de Dados
- Análise de Projetos Orientado a Objetos
- Programação Cliente Servidor
- Linguagens de Programação: C, C#, Pascal, PHP, ASP, Delphi, Java, JavaScript
- Programação WEB



Roteiro

- Palavras reservadas
- Tipos de dados
- Operadores
 - Aritméticos
 - Atribuição Compostos
 - Relacionais
 - Lógicos
 - Incremento e Decremento
- Estruturas de decisão
 - if
 - if-else
 - if-elseif-else
 - Switch
- Estruturas de Controle
 - For
 - While
 - Do – while
- Break e Continue
- Encerrando a Aplicação
- Vetores
- Classe Math



Palavras Reservadas

abstract	as	base
catch	char	checked
default	delegate	do
explicit	extern	false
foreach	goto	if
object	operator	out
public	readonly	ref
sizeof	stackalloc	static
throw	true	try
unsafe	ushort	using unibratesec.edu.br



Palavras Reservadas

bool	break	byte	case
class	const	continue	decimal
double	else	enum	event
finally	fixed	float	for
long	namespace	new	null
override	paramsw	private	protected
return	sbyte	sealed	short
string	struct	switch	this
typeof	uint	ulong	unchecked
virtual	void	volatile	while



Tipos de Dados Primitivos

Tipo de Dado	Descrição	Intervalo	Tamanho(bits)
bool	Booleano	true ou false	8
char	Caractere único	0 a 32.768	16
int	Números inteiros	-2.147.483.648 a 2.147.483.647	32
long	Números inteiros(intervalo maior)	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	64
float	Números ponto flutuante	$\pm 1.5 \times 10^{-45}$ a $\pm 7.9 \times 10^{38}$	32
double	Números ponto flutuante de precisão dupla	$\pm 5.0 \times 10^{-324}$ a $\pm 1.7 \times 10^{308}$	64
string	Sequência de caracteres	Não aplicável	Não aplicável



Tipos de Dados Primitivos

Tipo de Dado	Descrição	Intervalo	Tamanho(bits)
object	Referência para qualquer objeto	Não se aplica	
decimal	Valores numéricos	$\pm 1.0 \times 10^{-28}$ a $\pm 7.9 \times 10^{28}$	128
uint	Números inteiros positivos	0 a 4.294.967.295	32
ulong	Números inteiros positivos (intervalo maior)	0 a 18.446.744.073.709.551.615	64
short	Números inteiros intervalo menor	-32.768 a 32.767	16
ushort	Números inteiros positivos (intervalo menor)	0 a 65.535	16
byte	Números inteiros positivos (intervalo menor)	0 a 255	8
sbyte	Números inteiros (intervalo menor)	-128 a 127	8



Operadores

- Aritméticos
- Atribuição Compostos
- Relacionais
- Lógicos
- Incremento e Decremento



Operadores Aritméticos

Adição	+	int a = 2 + 2; // a = 4
Subtração	-	int b = 5 - 2; // b = 3
Multiplicação	*	int c = 2 * 5; // c = 10
Divisão	/	int d = 6 / 3; // d = 2
Resto	%	int e = 5 % 2; // e = 1



Precedência Op. Aritméticos

Operadores	Operação	Ordem de avaliação
()	Parenteses	Avaliados primeiro. Se os parênteses estiverem aninhados, a expressão no par mais interno será avaliada primeiro, se tiverem vários pares de parênteses “no mesmo nível”, (isto é, não aninhados), eles serão avaliados da esquerda para a direita.
*, /, %	Multiplicação, divisão e resto	Avaliados em segundo lugar, se houver vários desses operadores, eles serão avaliados da esquerda para a direita.
+, -	Adição e subtração	Avaliados por último. Se houver vários desses operadores, eles serão avaliados da esquerda para a direita.



Precedência Op. Aritméticos

y = a * x * x + b * x + c;

6

1

2

4

3

5

z = p * r % q + w / x - y;

6

1

2

4

3

5



Operadores de Atribuição Composta

Op	Descrição	Exemplo
<code>+=</code>	Atribuição composta de adição	<code>int a = 2;</code> <code>a += 2; // a = 4</code>
<code>-=</code>	Atribuição composta de subtração	<code>int b = 3;</code> <code>b -= 1; // b = 2</code>
<code>*=</code>	Atribuição composta de multiplicação	<code>int c = 2;</code> <code>c *= 3; // c = 6</code>
<code>/=</code>	Atribuição composta de divisão	<code>int d = 8;</code> <code>d /= 4; // d = 2</code>
<code>%=</code>	Atribuição composta de resto	<code>int e = 5;</code> <code>e %= 2; // e = 1</code>



Operadores Relacionais

Op	Descrição	Exemplo
<	Maior que	<code>inta = 3, b = 2; (a > b) true (b > a) false</code>
>	Menor que	<code>inta = 3, b = 2; (a < b) false (b < a) true</code>
>=	Maior ou igual	<code>int a = 3, b = 2, c = 2; (a >= b) true (a >= c) true</code>
<=	Menor ou igual	<code>int a = 3, b = 2, c = 2; (a <= b) false (a <= c) true</code>
==	Igualdade	<code>int a = 3, b = 2, c = 2; (a == b) false (a == c) true</code>
!=	Diferença	<code>int a = 3, b = 2, c = 2; (a != b) true (a != c) false</code>



Operadores Lógicos

Op	Descrição	Exemplo
&&	E(and)	<pre>int a = 3, b = 2, c = 2; (a == b) && (b == c) // false</pre>
	Ou (or)	<pre>int a = 3, b = 2, c = 2; (a == b) (b == c) // true</pre>
!	Negação (not)	<pre>bool a = true, b = false; !a // false !b // true</pre>



Operadores de Incremento e Decremento

Op	Descrição	Exemplo
++	pré-incremento,	<pre>int a = 3; int b = ++a; // a = 4 & b = 4</pre>
++	pós-incremento	<pre>int a = 3; int b = a++; // a = 4 & b = 3</pre>
--	pré-decremento	<pre>int a = 3; int b = --a; // a = 2 & b = 2</pre>
--	pós-decremento	<pre>int a = 3; int b = a--; // a = 2 & b = 3</pre>



Operadores de Incremento e Decremento

Exemplo adicional:

```
int a = 2, b = 3;  
int c = a++ * b + 1;    // a = 3, b = 3, c = 7  
int c = ++a * b + 1;    // a = 3, b = 3, c = 10
```



Estruturas de Decisão

- Comandos utilizados para desviar o fluxo de execução de uma rotina e/ou programa
- Podem ser do tipo:
 - **if**
 - **if-else**
 - **if-elseif-else**
 - **switch**
 - Só pode ser usado com tipos primitivos: **int**, **long**, **float**, **double**, **decimal**, **string**, **chare** **bool**



Estrutura de Decisão - if

- Sintaxe

```
if (<condição>)  
    comando;
```

```
if (<condição>)  
{  
    comando1;  
    ...  
    comandoN;  
}
```

- <condição> é avaliada. Se for verdadeiro (**true**), o(s) comando(s) são executados. Caso contrário, o fluxo do programa continua imediatamente após o comando (ou o bloco de comandos).



Estrutura de Decisão - if

```
bool result = true;
```

```
if (result == true)
```

```
    Console.WriteLine("O valor da variável é verdadeiro.");
```

```
if (result == true) {
```

```
    Console.WriteLine("O valor da variável é verdadeiro.");
```

```
}
```



Estrutura de Decisão – if - else

- Sintaxe

```
if (<condição>
    comando1;
else
    comando2;
```

```
if (<condição>)
{
    comando1;
    ...
    comandoN;
}
else
{
    comando1;
    ...
    comandoN;
}
```



Estrutura de Decisão – if - else

```
bool result = true;  
if (result == true)  
    Console.WriteLine("O valor da variável é verdadeiro.");  
else  
    Console.WriteLine(" O valor da variável é falso.");
```

```
if (result == true) {  
    Console.WriteLine("O valor da variável é verdadeiro.");  
} else {  
    Console.WriteLine(" O valor da variável é falso.");  
}
```



Estrutura de Decisão II

else

```
if (<condição1>)  
{  
    comando1;  
    ...  
    comandoN;  
}  
else if (<condição2>)  
{  
    comando1;  
    ...  
    comandoN;  
}  
...  
else if (<condiçãoN>)  
{  
    comando1;  
    ...  
    comandoN;  
}  
else  
{  
    comando1;  
    ...  
    comandoN;  
}
```



Estrutura de Decisão II

else

```
int result = 0;  
if (result == 0)  
    Console.WriteLine("O valor da variável é zero.");  
else if (result == 1)  
    Console.WriteLine(" O valor da variável é um.");  
else  
    Console.WriteLine(" O valor da variável é maior que um.");
```

```
if (result == 0) {  
    Console.WriteLine("O valor da variável é zero.");  
} else if (result == 1){  
    Console.WriteLine(" O valor da variável é um.");  
} else if (result == 2){  
    Console.WriteLine(" O valor da variável é dois.");  
} else {  
    Console.WriteLine(" O valor da variável é maior que dois.");  
}
```




Estrutura de Decisão Mais

exemplos

```
double vendas = 5000;  
double salario= 2500;  
double bonus;  
if((vendas >= 1000) && (vendas <= 5000)) {  
    bonus= salario* 0.05;  
    Console.WriteLine("Bônus: " + bonus);  
}
```



Estrutura de Decisão Mais

exemplos

```
int idade = 18;  
if((idade > 16 && idade < 18) || (idade > 65)) {  
    Console.WriteLine("voto facultativo");  
}
```



Estrutura de Decisão switch

```
switch (<condição>)  
{  
    case <constante1>:  
        comando1;  
        ...  
        comandoN;  
        break;  
    case <constante2>:  
        comando1;  
        ...  
        comandoN;  
        break;  
    ...  
    case <constanteN>:  
        comando1;  
        ...  
        comandoN;  
        break;  
    default:  
        comando1;  
        ...  
        comandoN;  
}
```



Estrutura de Decisão switch

- Utilização
 - Só pode ser usado com tipos primitivos: **int, long, float, double, decimal, string, char e bool**



Estrutura de Decisão switch

```
string estadoCivil= "Solteiro";  
switch (estadoCivil)  
{  
    case "Casado":  
        Console.WriteLine("Você está casado!");  
        break;  
    case "Divorciado":  
        Console.WriteLine("Você está divorciado!");  
        break;  
    default:  
        Console.WriteLine("Você está solteiro!");  
}
```



Estruturas de Repetição

- for
- while
- do... While



Estrutura de Repetição For

- Em <inicialização> ocorre a declaração e inicialização de uma ou mais variáveis
- Após ocorrer a <inicialização> é avaliada a <condição>. Se ela for verdadeira,

```
for (<inicialização>; <condição>; <exp. de iteração>)  
{  
    comando1;  
    ...  
    comandoN;  
}
```

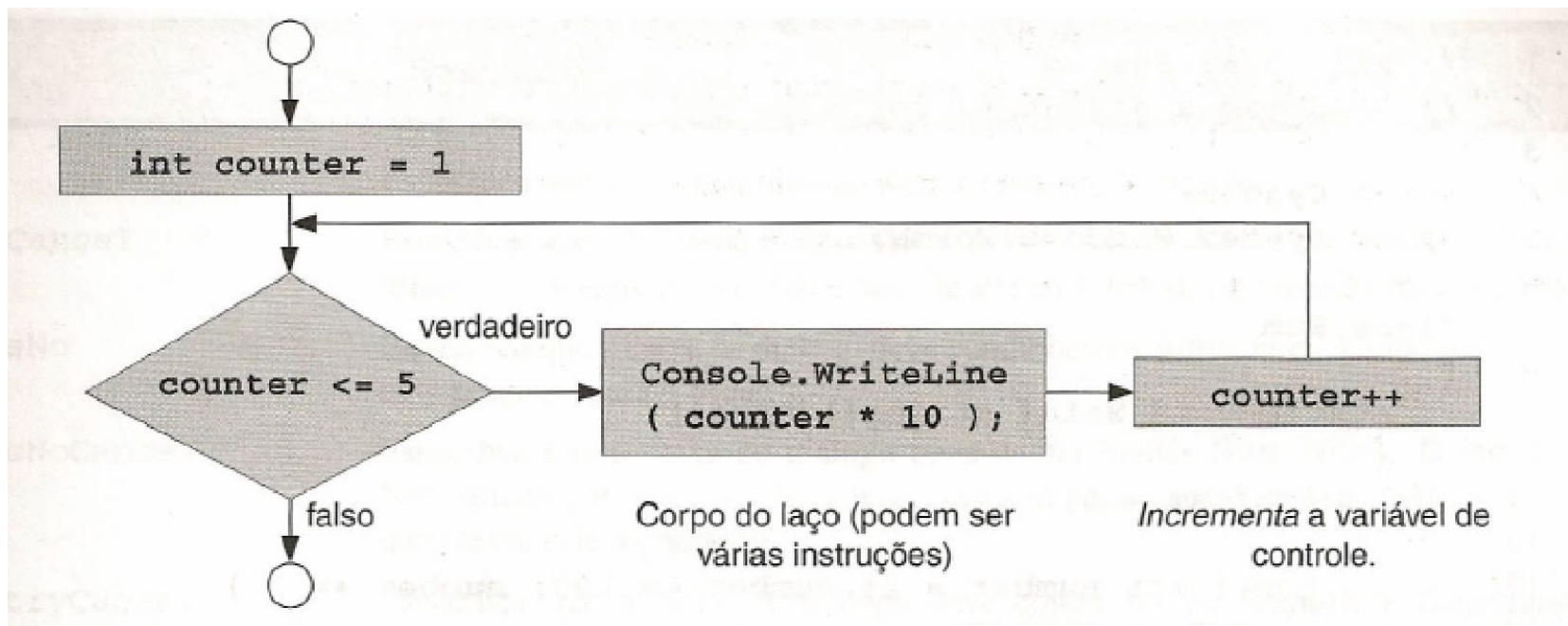


Estrutura de Repetição For

```
for( int i = 0; i < 10, i++){  
    Console.WriteLine("posição:" + i);  
}
```




Estrutura de Repetição For





ForEach

```
string[] lista = new string[10];  
foreach (var item in lista)  
{  
    Console.WriteLine(item);  
}
```



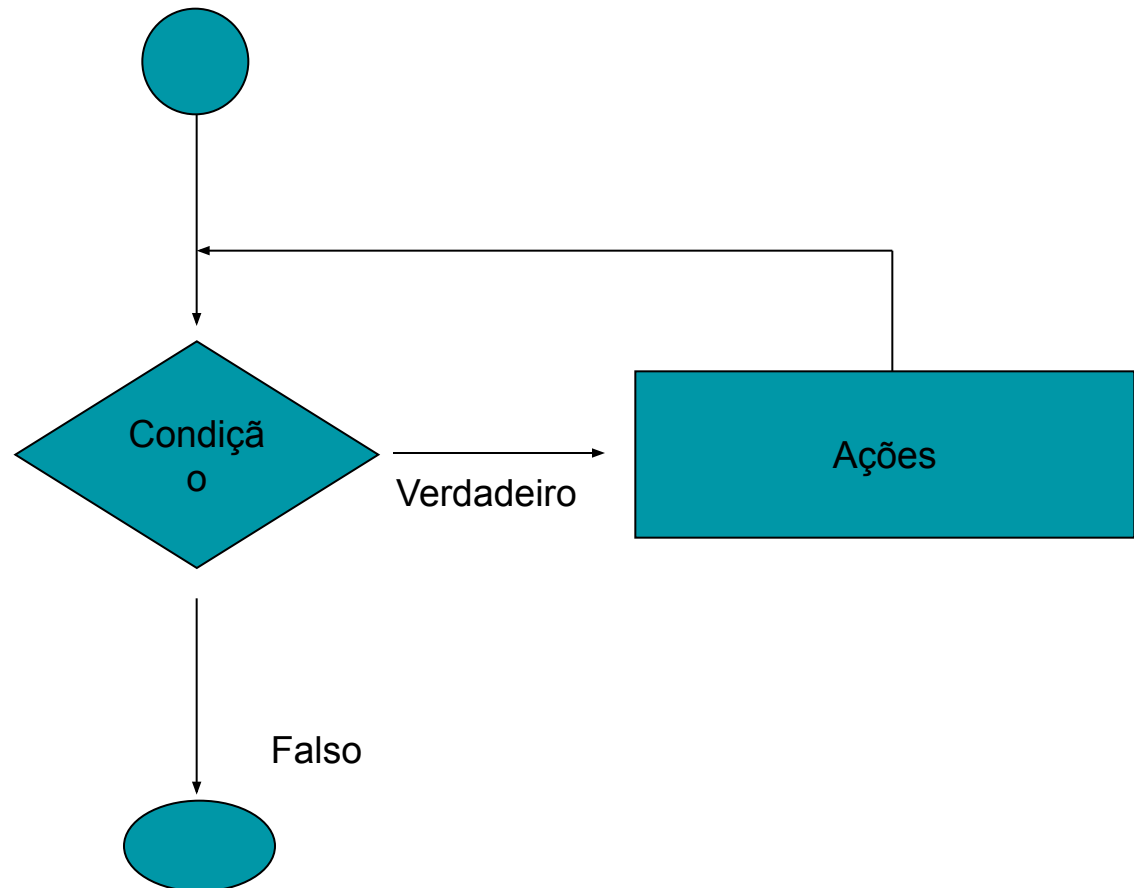
Estrutura de Repetição While

- O *nome* de uma *variável de controle* (ou contador de laço) usada para determinar se o laço prossegue sua execução.
- O *valor inicial* da variável de controle.
- O *incremento* (ou *decremento*) pelo qual a variável de controle é modificada a cada passagem no laço (também conhecida como *iteração do laço*).
- A condição que testa o *valor final* da variável de controle (isto é, se o laço deve continuar).

```
while (<condição>)  
{  
    comando1;  
    ...  
    comandoN;  
}
```



Estrutura de Repetição Do -While





Estrutura de Repetição While

```
int x = 0;  
while(i < 10){  
    Console.WriteLine("posição:" + i);  
    i++;  
}
```

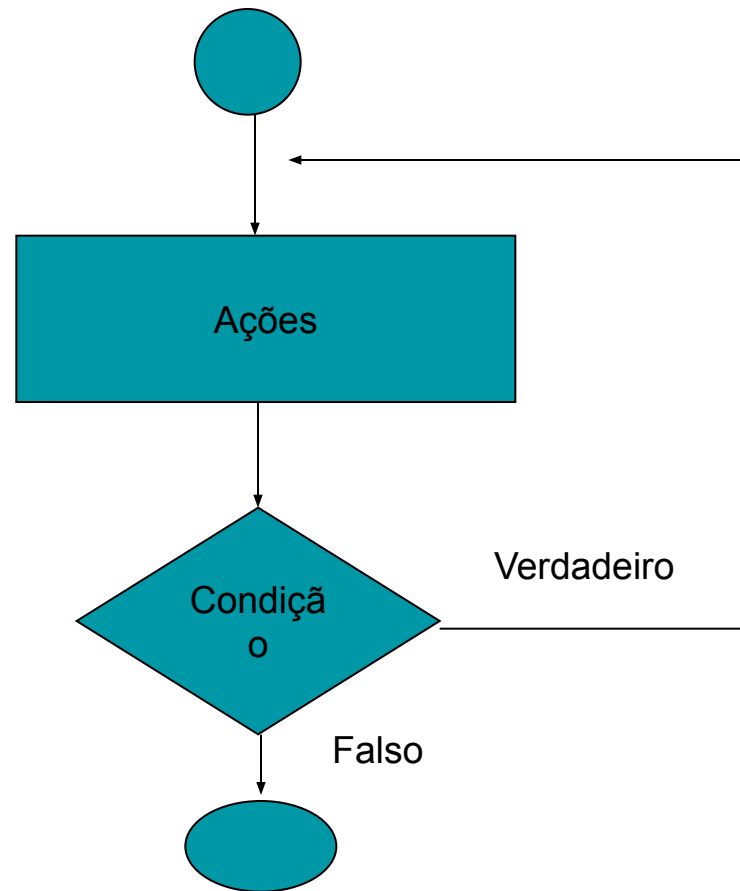


Estrutura de Repetição Do -While

- Semelhante a estrutura de repetição while, na qual o teste da condição de continuação do laço ocorre no início do laço, antes que o corpo seja executado.
- A estrutura do while testa a condição de continuação do laço após o corpo do laço ser executado.
- Portanto o corpo do laço é sempre executado pelo menos uma vez.



Estrutura de Repetição Do -While





Estrutura de Repetição Do -While

```
int a = 0;
```

```
do  
{  
    a++;  
    Console.WriteLine(a);  
} while (a < 10);
```




Break e Continue

- As instruções break e continue alteram o fluxo de controle.
- A instrução break quando utilizada nas estruturas de repetição for, while, do – while e switch, causa a saída imediata dessas estruturas.
- A instrução continue quando utilizada nas estruturas de repetição for, while, do – while e switch, pula as instruções do laço e passa para a próxima interação.



Exemplo break

```
for (int i = 0; i < 10; i++)  
    {  
        if (i == 6)  
        {  
            break;  
        }  
        Console.WriteLine(i);  
    }
```

//será impresso 0,1,2,3,4 e 5



Exemplo continue

```
for (int i = 0; i < 10; i++)  
{  
    if (i == 6)  
    {  
        continue;  
    }  
    Console.WriteLine(i);  
}
```

//será impresso 0,1,2,3,4,5,7,8 e 9. não será impresso o 6



Encerrando a Aplicação

- O comando para encerrar uma aplicação é:
`Environment.Exit(o);`



Vetores - Arrays

- Um array é um grupo de posições de memória adjacentes que têm o mesmo nome e tipo.
- São estruturas de dados lineares e estáticas.
- São estruturas de dados que armazenam valores homogêneos.
- Um vetor uma é estrutura de dados indexada,
- Vetores são, essencialmente, listas de informações de um mesmo tipo, armazenadas em posição contígua da memória, em ordem indexada.
- Um Vetor é uma das mais simples estruturas de dados.
- Tem seu tamanho pré-definido no momento de sua declaração ou criação.
- O seu tamanho não pode ser modificado durante o funcionamento do programa.



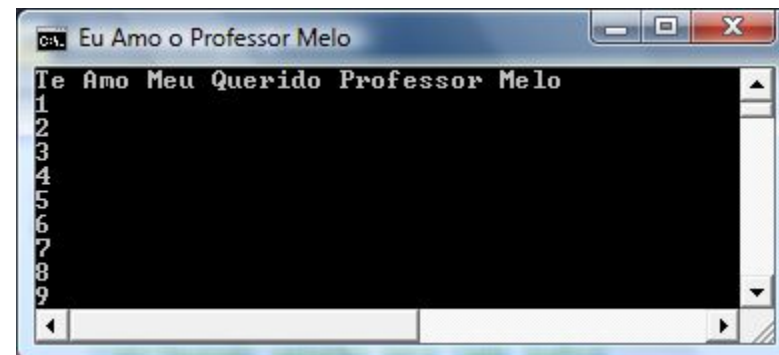
Vetores - Arrays

- Vetores são usados nos casos em que um conjunto de dados do mesmo tipo precisa ser armazenado em uma mesma estrutura, por exemplo:
 - o vetor Notas , que armazena o conjunto de notas da primeira avaliação é um conjunto de dados (ou valores) do tipo float;
 - o vetor Pares, que armazena os dez primeiros números naturais pares, é um vetor de valores do tipo int.



```
class Program
```

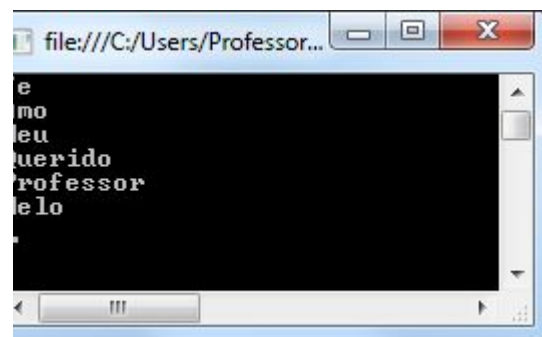
```
{  
    static void Main(string[] args)  
    {  
        string[] vetorString;  
        vetorString = new string[6];  
        //atribuindo valores para cada índice  
        vetorString[0] = "Te";  
        vetorString[1] = "Amo";  
        vetorString[2] = "Meu";  
        vetorString[3] = "Querido";  
        vetorString[4] = "Professor";  
        vetorString[5] = "Melo";  
        for (int i = 0; i < vetorString.Length; i++)  
        {  
            Console.Write(vetorString[i] + " ");  
        }  
        //atribuindo valores na sua declaração  
        int[] vetor = {1,2,3,4,5,6,7,8,9,0};  
        for (int i = 0; i < vetor.Length; i++)  
        {  
            Console.WriteLine(vetor[i]);  
        }  
        Console.ReadKey();  
    }  
}
```





```
static void Main(string[] args)
{
    string[] vetorString;
    vetorString = new string[6];
    vetorString[0] = "Te";
    vetorString[1] = "Amo";
    vetorString[2] = "Meu";
    vetorString[3] = "Querido";
    vetorString[4] = "Professor";
    vetorString[5] = "Melo";

    foreach (var item in vetorString)
    {
        Console.WriteLine(item);
    }
    Console.ReadKey();
}
```





Ordenando Vetor/Array

```
#region ordenando vetor
int[] vetor = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
//bubble sort - ordenação
for (int i = 0; i < vetor.Length; i++)
{
    for (int j = i; j < vetor.Length; j++)
    {
        //ordenar do maior para o menor, utilizar <(menor que)
        //ordenar do menor para o maior, utilizar >(maior que)
        if(vetor[i] > vetor[j]){
            int aux = vetor[i];
            vetor[i] = vetor[j];
            vetor[j] = aux;
        }
    }
}

for (int i = 0; i < vetor.Length; i++)
{
    //Console.WriteLine("Te amo melo " + (i+1) + " Vez(es)");
    Console.WriteLine(vetor[i]);
}

#endregion
```



Classe Math

- A classe Math permite que o programador efetue certos cálculos matemáticos comuns.



Nome	Descrição
<u>Abs(Decimal)</u>	Retorna o valor absoluto de um número <u>Decimal</u> .
<u>Abs(Double)</u>	Retorna o valor absoluto de um número de ponto flutuante de precisão dupla.
<u>Abs(Int16)</u>	Retorna o valor absoluto de um inteiro assinado de 16 bits.
<u>Abs(Int32)</u>	Retorna o valor absoluto de um inteiro assinado de 32 bits.
<u>Abs(Int64)</u>	Retorna o valor absoluto de um inteiro assinado de 64 bits.
<u>Abs(SByte)</u>	Retorna o valor absoluto de um inteiro assinado de 8 bits.
<u>Abs(Single)</u>	Retorna o valor absoluto de um número de ponto flutuante de precisão simples.
<u>Acos</u>	Retorna o ângulo cujo cosseno é o número especificado.
<u>Asin</u>	Retorna o ângulo cujo seno é o número especificado.
<u>Atan</u>	Retorna o ângulo cuja tangente é o número especificado.
<u>Atan2</u>	Returns the angle whose tangent is the quotient of two specified numbers.
<u>BigMul</u>	Produt o produto completo de dois números de 32 bits.
<u>Ceiling(Decimal)</u>	Retorna o menor valor de integral é maior que ou igual ao número de decimal especificado.
<u>Ceiling(Double)</u>	Retorna o menor valor de integral é maior ou igual ao número especificado de ponto flutuante de precisão dupla.
<u>Cos</u>	Retorna o cosseno do ângulo especificado.
<u>Cosh</u>	Retorna o cosseno hiperbólico do ângulo especificado.
<u>DivRem(Int32, Int32, Int32)</u>	Calcula o quociente de dois inteiros assinados de 32 bits e também retorna o resto de um parâmetro de saída.
<u>DivRem(Int64, Int64, Int64)</u>	Calcula o quociente de dois inteiros assinados de 64 bits e também retorna o resto de um parâmetro de saída.
<u>Exp</u>	Retorna e elevado à potência especificado.



Nome	Descrição
<u>Floor(Decimal)</u>	Retorna o maior inteiro menor ou igual ao número de decimais especificado.
<u>Floor(Double)</u>	Returns the largest integer less than or equal to the specified double-precision floating-point number.
<u>IEEERemainder</u>	Retorna o número especificado do resto que resulta da divisão de um número especificado por outro.
<u>Log(Double)</u>	Retorna o natural (base e) o logaritmo de um número especificado.
<u>Log(Double, Double)</u>	Retorna o logaritmo de um número especificado em uma base especificada.
<u>Log10</u>	Retorna o logaritmo de base 10 de um número especificado.
<u>Max(Byte, Byte)</u>	Retorna o maior dos dois inteiros não assinados de 8 bits.
<u>Max(Decimal, Decimal)</u>	Retorna o maior dos dois números decimais.
<u>Max(Double, Double)</u>	Retorna o maior dos dois números de ponto flutuante de precisão dupla.
<u>Max(Int16, Int16)</u>	Retorna o maior dos dois inteiros assinados de 16 bits.
<u>Max(Int32, Int32)</u>	Retorna o maior dos dois inteiros assinados de 32 bits.
<u>Max(Int64, Int64)</u>	Retorna o maior dos dois inteiros assinados de 64 bits.
<u>Max(SByte, SByte)</u>	Retorna o maior dos dois inteiros assinados de 8 bits.
<u>Max(Single, Single)</u>	Retorna o maior dos dois números de ponto flutuante de precisão simples.
<u>Max(UInt16, UInt16)</u>	Retorna o maior dos dois inteiros não assinados de 16 bits.
<u>Max(UInt32, UInt32)</u>	Retorna o maior dos dois inteiros não assinados de 32 bits.
<u>Max(UInt64, UInt64)</u>	Retorna o maior dos dois inteiros não assinados de 64 bits.



Nome	Descrição
<u>Min(Byte, Byte)</u>	Retorna o menor dos dois inteiros não assinados de 8 bits.
<u>Min(Decimal, Decimal)</u>	Retorna o menor dos dois números decimais.
<u>Min(Double, Double)</u>	Retorna o menor dos dois números de ponto flutuante de precisão dupla.
<u>Min(Int16, Int16)</u>	Retorna o menor dos dois inteiros assinados de 16 bits.
<u>Min(Int32, Int32)</u>	Retorna o menor dos dois inteiros assinados de 32 bits.
<u>Min(Int64, Int64)</u>	Retorna o menor dos dois inteiros assinados de 64 bits.
<u>Min(SByte, SByte)</u>	Retorna o menor dos dois inteiros assinados de 8 bits.
<u>Min(Single, Single)</u>	Retorna o menor dos dois números de ponto flutuante de precisão simples.
<u>Min(UInt16, UInt16)</u>	Retorna o menor dos dois inteiros não assinados de 16 bits.
<u>Min(UInt32, UInt32)</u>	Retorna o menor dos dois inteiros não assinados de 32 bits.
<u>Min(UInt64, UInt64)</u>	Retorna o menor dos dois inteiros não assinados de 64 bits.
<u>Pow</u>	Retorna um número especificado elevado à potência especificada.
<u>Round(Decimal)</u>	Arredonda um valor decimal para o valor inteiro mais próximo.
<u>Round(Double)</u>	Arredonda um valor de ponto flutuante de precisão dupla para o valor inteiro mais próximo.
<u>Round(Decimal, Int32)</u>	Arredonda um valor decimal para um número especificado de dígitos fracionários.
<u>Round(Decimal, MidpointRounding)</u>	Arredonda um valor decimal para o inteiro mais próximo. Um parâmetro especifica como arredondar o valor se for situadas entre dois outros números.
<u>Round(Double, Int32)</u>	Arredonda um valor de ponto flutuante de dupla precisão para um número especificado de dígitos fracionários.
<u>Round(Double, MidpointRounding)</u>	Arredonda um valor de ponto flutuante de precisão dupla para o inteiro mais próximo. Um parâmetro especifica como arredondar o valor se for situadas entre dois outros números.
<u>Round(Decimal, Int32, MidpointRounding)</u>	Arredonda um valor decimal para um número especificado de dígitos fracionários. Um parâmetro especifica como arredondar o valor se for situadas entre dois outros números.
<u>Round(Double, Int32, MidpointRounding)</u>	Arredonda um valor de ponto flutuante de dupla precisão como o número especificado de dígitos fracionários. Um parâmetro especifica como arredondar o valor se for situadas entre dois outros números.



Nome	Descrição
<u>Sign(Decimal)</u>	Retorna um valor indicando o sinal de um número decimal.
<u>Sign(Double)</u>	Retorna um valor indicando o sinal de um número de ponto flutuante de precisão dupla.
<u>Sign(Int16)</u>	Retorna um valor indicando o sinal de um inteiro assinado de 16 bits.
<u>Sign(Int32)</u>	Retorna um valor indicando o sinal de um inteiro assinado de 32 bits.
<u>Sign(Int64)</u>	Retorna um valor indicando o sinal de um inteiro assinado de 64 bits.
<u>Sign(SByte)</u>	Returns a value indicating the sign of an 8-bit signed integer.
<u>Sign(Single)</u>	Retorna um valor indicando o sinal de um número de ponto flutuante de precisão simples.
<u>Sin</u>	Retorna o seno do ângulo especificado.
<u>Sinh</u>	Retorna o seno hiperbólico do ângulo especificado.
<u>Sqrt</u>	Retorna a raiz quadrada de um número especificado.
<u>Tan</u>	Retorna a tangente do ângulo especificado.
<u>Tanh</u>	Retorna a tangente hiperbólica do ângulo especificado.
<u>Truncate(Decimal)</u>	Calcula a parte integrante de um número de decimal especificado.
<u>Truncate(Double)</u>	Calcula a parte integrante de um determinado número de ponto flutuante de precisão dupla.



Passagem de parâmetros por valor

- *Por valor:* Na passagem por valor, uma cópia do parâmetro é criada na pilha local do método, e mesmo que o parâmetro seja modificado pelo método, esta modificação não será visível fora dele.



Exemplo por Valor

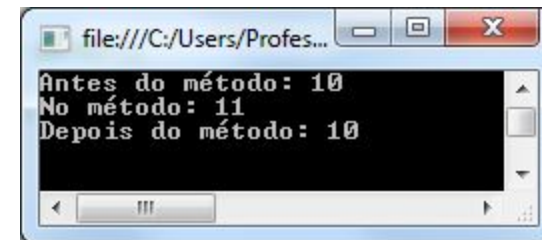
```
class Program
{
    static public void metRef(ref int parametro) ...

    static public void metValor( int parametro)
    {
        parametro++;
        Console.WriteLine("No método: " + parametro);
    }

    static void Main(string[] args)
    {
        int parRef = 10;

        por referencia

        #region por valor
        Console.WriteLine("Antes do método: " + parRef);
        metValor(parRef);
        Console.WriteLine("Depois do método: " + parRef);
        Console.ReadLine();
        #endregion
    }
}
```





por referência

- *Por referência: Se desejarmos que as modificações efetuadas no parâmetro sejam visíveis externamente, podemos passar o parâmetro por referência, e neste caso não apenas uma cópia local é criada, mas a referência do parâmetro na memória é passada e qualquer modificação que este sofrer no método será visível externamente. Para passar parâmetros por referência usamos a palavra reservada **ref** antes do tipo do parâmetro.*



Exemplo por Referência

```
class Program
{

    static public void metRef(ref int parametro) ...

    static public void metValor( int parametro) ...

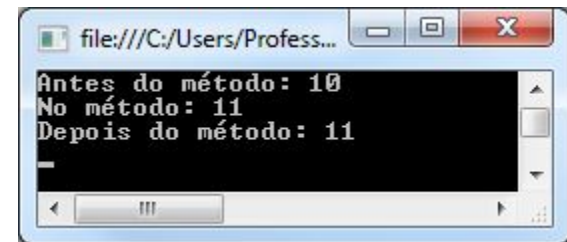
    static void Main(string[] args)
    {

        int parRef = 10;
        |
        #region por referencia

        Console.WriteLine("Antes do método: " + parRef);
        metRef(ref parRef);
        Console.WriteLine("Depois do método: " + parRef);
        Console.ReadLine();
        #endregion

        por valor

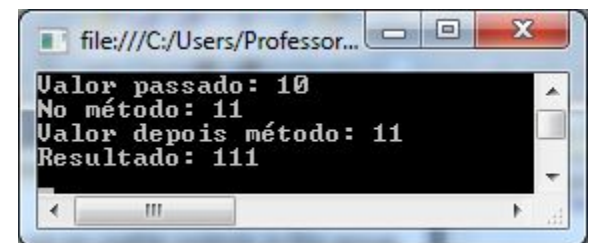
    }
}
```





Parâmetros de saída: OUT

- Existe um outro tipo de parâmetro por referência conhecido como parâmetro de saída. Esse tipo de parâmetro resolve o nosso problema porque ele não precisa ser inicializado para poder ser passado ao método, porque o seu valor inicial não tem utilidade, dado que a sua única finalidade é servir como valor de retorno.



```
file:///C:/Users/Professor...  
Valor passado: 10  
No método: 11  
Valor depois método: 11  
Resultado: 111
```



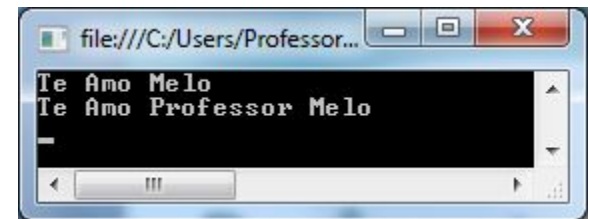
Parâmetros Facultativos

```
class Program
{
    #region

    static public void metParFacultativo(string nome="Melo")
    {
        Console.WriteLine("Te Amo " + nome);
    }

    static void Main(string[] args)
    {
        int parRef = 10;
        por referencia
        por valor
        out

        metParFacultativo();
        metParFacultativo("Professor Melo");
        Console.ReadKey();
    }
}
```





params

- O params palavra-chave permite que você especifique um parâmetro do método que leva um número variável de argumentos.



```
public static void UseParams(params int[] list)
{
    for (int i = 0; i < list.Length; i++)
    {
        Console.Write(list[i] + " ");
    }
    Console.WriteLine();
}
```

```
public static void UseParams2(params object[] list)
{
    for (int i = 0; i < list.Length; i++)
    {
        Console.Write(list[i] + " ");
    }
    Console.WriteLine();
}
```



```
static void Main()
{
    // You can send a comma-separated list of arguments of the
    // specified type.
    UseParams(1, 2, 3, 4);
    UseParams2(1, 'a', "test");

    // A params parameter accepts zero or more arguments.
    // The following calling statement displays only a blank line.
    UseParams2();

    // An array argument can be passed, as long as the array
    // type matches the parameter type of the method being called.
    int[] myIntArray = { 5, 6, 7, 8, 9 };
    UseParams(myIntArray);

    object[] myObjArray = { 2, 'b', "test", "again" };
    UseParams2(myObjArray);

    // The following call causes a compiler error because the object
    // array cannot be converted into an integer array.
    //UseParams(myObjArray);

    // The following call does not cause an error, but the entire
    // integer array becomes the first element of the params array.
    UseParams2(myIntArray);
}
```

```
/*
Output:
    1 2 3 4
    1 a test

    5 6 7 8 9
    2 b test again
    System.Int32[]
*/
```




Tipos Enumerados

- Tipos enumerados são listas de valores constantes cuja representação interna se dá através de números inteiros. Cada constante definida num tipo enumerado mapeia um número inteiro específico, que começa pelo valor inteiro zero. Entretanto, este valor inicial pode ser sobrescrito quando assim especificado na declaração do tipo enumerado.
- Os membros de um tipo enumerado não possuem modificadores de acesso, e estes são acessíveis desde que o tipo enum que os contém seja acessível.



Tipos Enumerados

```
class Program
{
    enum meses
    {
        janeiro = 31,
        fevereiro = 28,
        marco = 31,
        abril = 30,
        maio = 31,
        junho = 30,
        julho = 31,
        agosto = 31,
        setembro = 30,
        outubro = 31,
        novembro = 30,
        dezembro = 31
    }

    #region

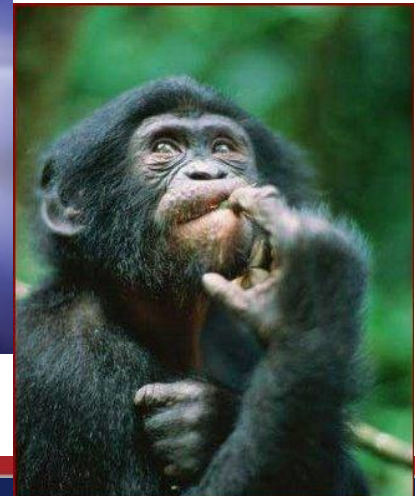
    static void Main(string[] args)
    {
        variaveis
        por referencia
        por valor
        out

        #region
        Console.WriteLine(meses.fevereiro);
        Console.WriteLine((int) meses.fevereiro+ "");
        Console.ReadKey();
    }
}
```





Perguntas





Referências

Bibliografia Básica (títulos, periódicos, etc.)						
Título/Periódico	Autor	Edição	Local	Editora	Ano	LT/BV¹
Microsoft Visual C# 2008: passo a passo	SHARP, John	1ª	Porto Alegre	Bookman	2008	LT
Desenvolvimento em camadas com C# .Net	CAMACHO JÚNIOR, Carlos Olavo de Azevedo	1ª	Florianópolis	Visual Books	2008	LT
C#: como programar	DEITEL, H. M.	1ª	São Paulo	Pearson	2003	LT
Bibliografia Complementar (títulos, periódicos, etc.)						
Título/Periódico	Autor	Edição	Local	Editora	Ano	LT/BV¹
Use a cabeça C#	STELLMAN, Andrew; GREENE, Jeniffer	2ª	Rio de Janeiro	Alta Books	2011	LT
Treinamento avançado em XML	FARIA, Rogério Amorim De	1ª	São Paulo	<u>Digerati</u>	2005	LT
Projeto de Banco de Dados com XML	GRAVES, Mark	1ª	São Paulo	Pearson	2003	BV