

Guia de Consulta Rápida

C#

Joel Saade

Novatec

Copyright © 2008 da Novatec Editora Ltda.

Todos os direitos reservados e protegidos pela Lei 9610 de 19/02/1998. É proibida a reprodução desta obra, mesmo parcial, por qualquer processo, sem prévia autorização, por escrito, do autor e da Editora.

Editor: RUBENS PRATES

ISBN: 978-85-7522-146-4

1ª edição: Dezembro/2007

NOVATEC EDITORA LTDA.

Rua Luís Antônio dos Santos, 110

02460-000 São Paulo SP – Brasil

Tel.: +55 11 6959-6529

Fax: +55 11 6950-8869

E-mail: novatec@novatec.com.br

Site: www.novatec.com.br

Conceitos básicos.....	5
Estrutura de um programa C#	5
Compilação e execução de programas	5
Identificadores.....	5
Variáveis	6
Case sensitivity.....	6
Comentários	7
Tipos de dados.....	7
Constantes.....	11
Seqüências de escape	12
Operadores.....	13
De atribuição	13
Aritméticos	13
Aritméticos de atribuição	13
Aritméticos: incremento e decremento	13
Condicional ternário	14
Bitwise.....	14
Comandos	17
Bloco de comandos	17
Decisão	18
Desvio	19
Iteração.....	21
Arrays	24
Arrays unidimensionais.....	25
Arrays bidimensionais.....	25
Array de strings.....	26
Manipulação de arrays com métodos	26
Classes e objetos	32
Estrutura de uma classe.....	32
Criação de objetos.....	33
Atribuição a variáveis do tipo reference	34
Construtor de classes.....	34
Destruitor de classes.....	35
Sobrecarga de construtores	36
Classes static.....	36
this	37
Herança de classes.....	38
Métodos virtuais	43
Classes abstratas e métodos abstratos.....	44
Classes sealed.....	45
Indexadores.....	46
Propriedades	47
Interfaces	48
Estruturas.....	50
Atribuição de estruturas	52
Array de estruturas	52
Métodos	53
Passagem de argumentos	53
Retorno de objetos	56
Sobrecarga de métodos.....	57
Métodos recursivos.....	57
Métodos e variáveis static	58
Delegates	58
Sobrecarga de operadores	59

Sobrecarga de operadores unários	60
Sobrecarga de operadores binários	60
Passagem de argumentos a um programa	61
Formatação de valores	62
Especificadores de formato para dados numéricos	63
Formatação com o método ToString()	63
Formatação de data e hora	64
Exceções	64
Tratamento de exceções	64
Múltiplos blocos catch	65
Captura de qualquer exceção	66
Bloco finally	66
Disparo de exceções	67
Propriedades da classe Exception	68
Estruturas dos tipos value predefinidos	68
Estrutura Boolean (bool)	68
Estrutura Char (char)	70
Estrutura Byte (byte)	74
Estrutura Int16 (short)	75
Estrutura Int32 (int)	76
Estrutura Int64 (long)	77
Estrutura SByte (sbyte)	78
Estrutura UInt16 (ushort)	80
Estrutura UInt32 (uint)	81
Estrutura UInt64 (ulong)	82
Estrutura Decimal (decimal)	84
Estrutura Double (double)	87
Estrutura Single (float)	89
Namespaces	91
Declaração de namespaces	91
Diretiva using	93
Namespaces aninhados	93
Modo unsafe	94
Ponteiros	94
Operador de endereço	95
Operador de de-referência	95
Acesso a membros de estruturas com ponteiros	95
Ponteiros e strings	96
Operador sizeof	96
Windows Forms	97
Comentários XML	99
Tags	99
Coleções genéricas	101
Coleção Dictionary<TK,TV>	101
Coleção LinkedList<T>	104
Coleção List<T>	107
Coleção Queue<T>	113
Coleção SortedDictionary<TK,TV>	115
Coleção SortedList<TK,TV>	119
Coleção Stack<T>	122
Opções do compilador	124
Palavras reservadas	126
Convenções/siglas	126
Índice remissivo	127

Conceitos básicos

Estrutura de um programa C#

Um programa C# é composto de uma classe e seus métodos. Mas podem estar presentes também estruturas, enumerações etc.

Exemplo:

```
(1) // estrutura de um programa C#
(2) using System;
(3) class Principal
(4) {
(5)     public static void Main()
(6)     {
(7)         Console.WriteLine("Hello World!");
(8)     }
(9) }
```

Linha	Descrição
1	Comentário, indicado pelo par de caracteres //.
2	Utilização do namespace <code>System</code> .
3	Declaração de classe, no caso, da classe <code>Principal</code> .
4	{, indica o início da classe <code>Principal</code> .
5	<code>Main()</code> , método obrigatório, em que a execução do programa é iniciada e terminada. <code>Main()</code> deve ser um método <code>static</code> e estar em uma classe ou estrutura.
6	{, indica o início do método <code>Main()</code> .
7	Exibe o string Hello World! , com salto de linha, na saída-padrão, utilizando o método <code>WriteLine</code> da classe <code>Console</code> . Sem o namespace <code>System</code> qualifica-se a classe para indicar o namespace a que pertence: <code>System.Console.WriteLine(...)</code> .
8	}, indica o fim do método <code>Main()</code> .
9	}, indica o fim da classe <code>Principal</code> .

Compilação e execução de programas

Programas C# são digitados em um editor de textos ou no IDE do Visual Studio. Compilação e execução podem ser feitas no IDE ou na linha de comando. A extensão de programas-fonte é `.cs` e o compilador é o programa **csc.exe**. Para usar a linha de comando, acesse o botão Iniciar: Microsoft Visual Studio 2005 | Visual Studio Tools | Visual Studio 2005 Command Prompt.

Para compilar programas na linha de comando, a sintaxe é:

```
csc programa.cs
```

Cria o programa `programa.exe`.

Identificadores

São nomes atribuídos pelo programador a variáveis, métodos etc. Devem obedecer às seguintes regras:

- Caracteres permitidos: alfabéticos (maiúsculos e minúsculos), numéricos e o `_` (sublinhado).
- O primeiro caractere deve ser alfabético ou o caractere `_`.
- Não pode ser uma palavra reservada. Mas se uma palavra reservada for iniciada pelo caractere “@”, esta poderá ser usada como um identificador. Por exemplo, `int @new;`

Variáveis

Variável refere-se a uma posição de memória em que um valor é armazenado, com possibilidade de ser alterado. Toda variável tem nome e tipo e deve ser declarada antes de ser utilizada.

Exemplos:

- (1) `int i;`
- (2) `int j = 99;`
- (3) `float saldo, valor, taxa;`
- (4) `double desconto = 900.5, saldo;`
- (5) `int total = p1 + p2;`

Linha	Comentário
1	Variável não inicializada.
2	Variável inicializada.
3	Variáveis de um mesmo tipo separadas por “,”.
4	Variáveis do mesmo tipo, apenas uma inicializada.
5	Variável inicializada com o valor de uma expressão.

Escopo e existência de variáveis

Variáveis podem ser declaradas em qualquer bloco. Um bloco define um escopo. Um escopo determina quais objetos são visíveis a outras partes de um programa e também a existência destes objetos. No escopo de um método, incluem-se também os seus argumentos (argumentos formais), caso existam.

Como regra geral, variáveis declaradas em um escopo não são visíveis (acessíveis) fora deste escopo. Os escopos podem ser aninhados: escopo contido em outro. Neste caso, o escopo externo engloba o escopo interno. Quando um bloco é iniciado, as suas variáveis são criadas (ocorre alocação de memória), quando o bloco termina, as variáveis deixam de existir (a memória alocada é liberada).

```
// escopo.cs
using System;
class Principal {
    public static void Main() {
        int VarM; // no escopo do método Main()
        VarM = 20;
        Console.Clear();
        if (VarM == 20)
        {
            int VarB = 30; // no escopo do bloco
            Console.WriteLine("VarM = " + VarM); // 20
            Console.WriteLine("VarB = " + VarB); // 30
        }
        //VarB = 40; // erro: fora do escopo do bloco
    }
}
```

Case sensitivity

C# distingue caracteres alfabéticos minúsculos dos respectivos maiúsculos, inclusive em relação aos seus comandos. Assim, as variáveis VALOR e valor são consideradas distintas.

Comentários

Servem para documentar o programa, sendo ignorados pelo compilador. Há dois tipos de comentários.

Tipo	Descrição
//	Comentário de linha única. Pode ser utilizado em uma linha de comando também.
/* ... */	Comentário multilinha.

Exemplos:

```
// comentário de linha única
int limite = 10;    // variável de controle
/* linha de comentário 1
   linha de comentário 2
*/
```

Tipos de dados

Há duas categorias de tipos de dados: **value** e **reference**.

Categoria	Descrição
value	Tipo cujas variáveis armazenam o próprio valor.
reference	Tipo cujas variáveis armazenam uma referência (endereço) para um valor e não este valor.

Tipos value predefinidos

bool

Permite a representação dos valores lógicos true e false.

Exemplo:

```
bool status = true;
```

char (caractere)

É um inteiro de 16 bits sem sinal com intervalo de valores entre 0 e 65.535 para representar caracteres Unicode.

Exemplos:

```
char letra = 'C', especial = '#';
```

decimal

Permite a representação de valores que podem ter uma parte fracionária. Útil em cálculos que envolvem valores monetários e ao contrário tipo ponto-flutuante não apresenta erros de arredondamento. Possui 128 bits com um intervalo de valores entre 1E-28 a 7.9E+28 e com 28 ou 29 dígitos significativos.

Exemplo:

```
decimal valor = 85.95m;
```

Escrever o caractere `m` ou `M` em uma constante `decimal`.

int (inteiro)

Permite a representação de valores inteiros. É dividido nos subtipos: `byte`, `long`, `sbyte`, `short`, `uint`, `ulong` e `ushort`.

Tipo/Subtipo	Descrição
int	Inteiro com sinal.
uint	Inteiro sem sinal.
byte	Inteiro sem sinal.
sbyte	Inteiro com sinal.
short	Inteiro curto com sinal.
ushort	Inteiro curto sem sinal.
long	Inteiro longo com sinal.
ulong	Inteiro longo sem sinal.

Exemplos:

byte contador = 0;

short limite = -1;

Tabela do tipo integer e seus subtipos

Tipo	Comprimento (em bits)	Intervalo de valores
int	32	-2.147.483.648 a 2.147.483.647
uint	32	0 a 4.294.967.295
byte	8	0 a 255
sbyte	8	-128 a 127
short	16	-32.768 a 32.767
ushort	16	0 a 65.535
long	64	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
ulong	64	0 a 18.446.744.073.709.551.615

ponto-flutuante

Permite a representação de valores que podem conter uma parte fracionária. É dividido em: float e double. Além da notação comum (decimal) é possível escrever valores ponto-flutuante em notação exponencial, com o caractere E ou e, seguido de um expoente positivo (+) ou negativo (-) de uma base 10 implícita. Se o expoente for positivo, o sinal + poderá ser omitido.

Exemplos:

float saldo = 1900F; // notação comum (decimal)

double divida = 7.5E3; // notação exponencial. 7.5×10^3

Escrever o caractere f ou F em uma constante float.

Tabela do tipo ponto-flutuante

Tipo	Comprimento (em bits)	Intervalo de valores	Precisão (dígitos)
float	32	1.5E-45 a 3.4E+38	7
double	64	5E-324 a 1.7E+308	15 ou 16

Tipos value definidos pelo usuário

enum (enumeração)

Conjunto de valores constantes identificados por nomes, chamado lista de enumeração. Toda enumeração tem um tipo-base associado e o conjunto de valores é desse tipo.

```
enum nome [: tipo] { lista } [;]
```

Elemento	Descrição
<i>nome</i>	Nome da enumeração.
<i>tipo</i>	Tipo-base. Pode ser inteiro e seus subtipos (default = int).
<i>lista</i>	Lista de enumeração, composta de constantes separadas por “,”.

Exemplo: Declara um tipo enum chamado cores

```
enum cores : byte {azul, verde, vermelho, amarelo};
```

Cada constante da lista tem um valor associado: o valor da primeira constante, por default, é 0. Então, azul tem o valor 0, verde tem o valor 1, e assim por diante. O valor da constante seguinte é maior em uma unidade que o valor da constante anterior.

É possível inicializar a primeira constante da lista de enumeração, alterando o valor default, como em:

```
enum cores {azul = 1, verde, vermelho, amarelo};
```

A constante verde terá o valor 2, vermelho, o valor 3, e assim por diante.

Embora uma constante da lista de enumeração possa ser utilizada onde um inteiro pode ser utilizado, é preciso um type cast para converter um tipo enum em inteiro.

```
// type cast do tipo enum para o tipo int
// enum01.cs
using System;
class Principal {
    enum cores {azul,verde,vermelho,amarelo};
    public static void Main() {
        int cor1;
        cor1 = (int)cores.azul;
        Console.Clear();
        Console.WriteLine("cor1 tem o valor " + cor1);
        Console.WriteLine(cores.verde + " tem o valor " +
            (int) cores.verde);
    }
}
```

As constantes da lista de enumeração são acessadas pelo operador “.”, como em cores.azul. E como uma enumeração é um tipo, então é possível declarar variáveis desse tipo.

```
// variáveis do tipo enumeração
// enum02.cs
using System;
class Principal {
    enum cores {azul,verde,vermelho,amarelo};
    public static void Main() {
        cores ICor; // variável do tipo enumeração
        Console.Clear();
        for(ICor = cores.azul;ICor <= cores.amarelo;++ICor)
            Console.WriteLine(ICor + " - valor " + (int)ICor);
    }
}
```

Conversões de tipos, em atribuições

Para os tipos predefinidos existem conversões predefinidas. Há dois tipos de conversão: implícita (ou automática) e explícita.

Conversão	Descrição
implícita	Feita pelo compilador, desde que seja segura.
explícita	Feita pelo usuário, forçando a conversão de um tipo em outro (type cast).

Exemplo: Conversão implícita, do tipo `int` para o tipo `long`.

```
int ValorInt = 98;
long ValorLong = ValorInt;
Console.WriteLine("Valor long: " + ValorLong); // 98
```

Conversão segura: o tipo `int` tem um intervalo de valores menor que o do tipo `long`.

Type cast

Conversão de um tipo em outro, explicitamente. É usado nos casos em que não há uma conversão implícita. Veja **Tabela de conversões numéricas implícitas**, pág. 10.

(*tipo*) *valor*

Elemento	Descrição
<i>tipo</i>	Tipo no qual <i>valor</i> será convertido.
<i>valor</i>	Valor a ser convertido em <i>tipo</i> .

Exemplo 1: Type cast, do tipo `int` para o tipo `short`.

```
int ValInt = 85000;
short ValShort = (short)ValInt; // type cast
Console.WriteLine("Valor short: " + ValShort);
```

Overflow: o tipo `short` tem um intervalo de valores menor que o do tipo `int`.

Exemplo 2: Type cast, do tipo `int` para o tipo `short`, seguro.

```
int ValInt = 85;
short ValShort = (short)ValInt; // type cast
Console.WriteLine("Valor short: " + ValShort);
```

Embora o tipo `short` tenha um intervalo de valores menor que o do tipo `int`, o valor 85 pode ser armazenado.

As conversões implícitas são conforme a tabela a seguir. Para as conversões não constantes da mesma, utiliza-se o `type cast`.

Tabela de conversões numéricas implícitas

De	Para
<code>byte</code>	<code>short</code> , <code>ushort</code> , <code>int</code> , <code>uint</code> , <code>long</code> , <code>ulong</code> , <code>float</code> , <code>double</code> , <code>decimal</code> .
<code>char</code>	<code>ushort</code> , <code>int</code> , <code>uint</code> , <code>long</code> , <code>ulong</code> , <code>float</code> , <code>double</code> , <code>decimal</code> .
<code>float</code>	<code>double</code> .
<code>int</code>	<code>long</code> , <code>float</code> , <code>double</code> , <code>decimal</code> .
<code>long</code>	<code>float</code> , <code>double</code> , <code>decimal</code> .
<code>sbyte</code>	<code>short</code> , <code>int</code> , <code>long</code> , <code>float</code> , <code>double</code> , <code>decimal</code> .
<code>short</code>	<code>int</code> , <code>long</code> , <code>float</code> , <code>double</code> , <code>decimal</code> .
<code>uint</code>	<code>long</code> , <code>ulong</code> , <code>float</code> , <code>double</code> , <code>decimal</code> .
<code>ulong</code>	<code>float</code> , <code>double</code> , <code>decimal</code> .
<code>ushort</code>	<code>int</code> , <code>uint</code> , <code>long</code> , <code>ulong</code> , <code>float</code> , <code>double</code> , <code>decimal</code> .

Tipos reference predefinidos

object

C# define a classe `object`, que é a classe-base para todas as outras classes e para todos os outros tipos. Assim, uma variável do tipo `object` pode referenciar um objeto de qualquer outro tipo.

string

Tipo que armazena caracteres Unicode. Constantes string são escritas entre aspas. É possível usar seqüências de escape.

Exemplo 1: Constante string.

```
Console.WriteLine("Eu sou um string");
```

Exemplo 2: Constante string com seqüência de escape.

```
Console.WriteLine("Caixa d'água"); // Caixa d'água
```

Há também a **constante string verbatim**, que é iniciada pelos caracteres `@`. O seu conteúdo é tratado como é e pode ter mais de uma linha. Assim, caracteres de tabulação e outros podem ser incluídos sem a utilização das seqüências de escape. No meio de uma constante, duas “ consecutivas equivalem a uma.

Exemplo 1:

```
Console.WriteLine(@"Sant"Ana"); // Sant"Ana
```

Exemplo 2:

```
Console.WriteLine(@"Constante  
string  
verbatim");
```

Assume-se um **Enter** no final da primeira e segunda linhas.

Comparação de strings

Os operadores relacionais `==` e `!=` comparam os conteúdos de strings e não referências.

Exemplo:

```
string str1 = "C#", str2 = "C#";  
if (str1 == str2)  
    Console.WriteLine("Strings iguais);
```

Concatenação de strings

A concatenação de strings é feita pelo operador `+`.

Exemplo:

```
string str1 = "Microsoft ", string str2 = "C#";  
string str3 = str1 + str2;  
Console.WriteLine(str3); // Microsoft C#
```

Constantes

Valores que não se alteram em um programa.

Numéricas

As constantes numéricas inteiras e de ponto-flutuante, por default, são assim tratadas: as inteiras, como `int`, `uint`, `long` ou `ulong`, conforme os seus valores. E as de ponto-flutuante, como `double`. Pode-se mudar essa regra por meio de um sufixo.

Sufixo	Descrição
l, L	Constante do tipo <code>long</code> .
u, U	Constante do tipo <code>uint</code> .
ul, UL	Constante do tipo <code>ulong</code> .
f, F	Constante do tipo <code>float</code> .
m, M	Constante do tipo <code>decimal</code> .

Exemplos:

```
Console.WriteLine(2000L); // constante long
float saldo1 = 1900.0F; // constante float
```

As constantes inteiras podem ser escritas no sistema hexadecimal, com o prefixo `0x` ou `0X`.

Exemplo:

```
int limite = 0xF; // equivale ao decimal 15
```

Simbólicas

São valores constantes identificados por um nome.

```
const tipo nome = valor;
```

Elemento	Descrição
<i>tipo</i>	Tipo da constante simbólica.
<i>nome</i>	Nome da constante simbólica.
<i>valor</i>	Valor da constante simbólica.

Exemplo:

```
const double taxa = 5.5;
const int limite = 10;
int[] valores = new int[limite];
```

Seqüências de escape

Uma seqüência de escape é formada pelo caractere `\` seguido de um ou mais caracteres, que permitem o uso de caracteres que não podem ser digitados ou que têm um significado especial em C#.

Tabela de seqüências de escape

Seqüência	Descrição
<code>\a</code>	Emite um alerta sonoro.
<code>\b</code>	Move o cursor para a esquerda.
<code>\f</code>	Salta para uma nova página.
<code>\n</code>	Salta de linha e posiciona o cursor no seu início.
<code>\r</code>	Move o cursor para o início da linha.
<code>\t</code>	Move o cursor para a próxima posição de tabulação horizontal.
<code>\v</code>	Move o cursor para a próxima posição de tabulação vertical.
<code>\"</code>	O caractere <code>"</code> (aspa).
<code>\'</code>	O caractere <code>'</code> (apóstrofo).
<code>\\</code>	O caractere <code>\</code> .
<code>\0</code>	O caractere nulo.
<code>\xnnn</code>	O caractere correspondente ao valor hexadecimal <i>nnn</i> .

```
// escape.cs
using System;
```

```

class Principal {
    public static void main() {
        // seqüências de escape hexadecimais
        char car1 = '\x043';
        char car2 = '\x023';
        Console.Clear();
        Console.Write(car1);           // C
        Console.Write(car1 + " " + car2); // C #
    }
}

```

Operadores

De atribuição

Operador	Descrição
=	Atribui valor a uma variável, propriedade ou indexador.

Exemplos:

```

int a;
a = 1;           // atribuição única
int b,c,d;
// atribuição encadeada. As três variáveis terão o valor 9
b = c = d = 9;

```

Aritméticos

Operador	Descrição
*	Multiplicação.
/	Divisão.
%	Resto da divisão inteira ou ponto-flutuante.
+	Adição.
-	Subtração.

Aritméticos de atribuição

Operador	Descrição
*=	Multiplicação com atribuição.
/=	Divisão com atribuição.
%=	Resto da divisão inteira ou ponto-flutuante com atribuição.
+=	Adição com atribuição.
-=	Subtração com atribuição.

Exemplo:

```

x+= 2;   equivale à x = x + 2;
y-= 1;   equivale à y = y - 1;

```

Aritméticos: incremento e decremento

Operador	Descrição
++	Adiciona 1 ao seu operando.
--	Subtrai 1 do seu operando.

Formas de utilização:

Forma	Descrição
<code>++operando</code>	Adiciona 1 ao valor do <i>operando</i> e então utiliza este novo valor.
<code>operando++</code>	Utiliza o valor corrente do <i>operando</i> e então adiciona 1 a este valor.
<code>--operando</code>	Subtrai 1 do valor do <i>operando</i> e então utiliza este novo valor.
<code>operando--</code>	Utiliza o valor corrente do <i>operando</i> e então subtrai 1 deste valor.

Exemplo 1: Operador ++, na forma de sufixo.

```
int x = 6; int y = 0;
y = x++;
Console.WriteLine("x vale " + x); // 7
Console.WriteLine("y vale " + y); // 6
```

Exemplo 2: Operador ++, na forma de prefixo.

```
int x = 6; int y = 0;
y = ++x;
Console.WriteLine("x vale " + x); // 7
Console.WriteLine("y vale " + y); // 7
```

Condicional ternário

?:

É uma forma alternativa a um comando `if ... else`.

```
exp1 ? exp2 : exp3;
```

Se *exp1* for verdadeira, o valor de toda a expressão será o valor de *exp2*, caso contrário, o valor de *exp3*.

Exemplo:

```
int x = -4, y;
y = x < 0 ? -x : x;
Console.WriteLine("y = " + y); // 4
```

Bitwise

São operadores que manipulam os bits de valores inteiros.

Bitwise de deslocamento

Os operadores bitwise de deslocamento são: `<<` e `>>`.

`<<`

Deslocamento de bits à esquerda. Válido para os tipos `int`, `uint`, `long` e `ulong`. As posições de bits desocupadas (mais à direita) serão preenchidas com 0 e os bits que passarem do limite (mais à esquerda), serão perdidos. Se *valor* for do tipo `int` ou `uint`, a quantidade de bits a ser deslocada será dada pelos 5 bits mais à direita de *n*. Se *valor* for do tipo `long` ou `ulong`, a quantidade de bits a ser deslocada será dada pelos 6 bits mais à direita de *n*.

```
valor << n
```

Elemento	Descrição
<i>valor</i>	Valor a ter seus bits deslocados para a esquerda.
<i>n</i>	Número de bits a ser deslocado. Deve ser do tipo <code>int</code> .

Exemplo:

```
uint valor = 12;
valor = valor << 1;
Console.WriteLine("valor: " + valor); // 24
```

>>

Deslocamento de bits à direita. Válido para os tipos `int`, `uint`, `long` e `ulong`. Se *valor* for do tipo `int` ou `uint`, a quantidade de bits a ser deslocada será dada pelos 5 bits mais à direita de *n*. Se *valor* for do tipo `long` ou `ulong`, a quantidade de bits a ser deslocada será dada pelos 6 bits mais à direita de *n*. Se *valor* for do tipo `int` ou `long`, os bits mais à esquerda desocupados terão o valor do bit de sinal e os que passarem do limite (mais à direita), serão perdidos (um deslocamento aritmético). Se *valor* for do tipo `uint` ou `ulong`, os bits mais à esquerda serão preenchidos com 0 e os que passarem do limite (mais à direita), serão perdidos (um deslocamento lógico).

valor >> *n*

Elemento	Descrição
----------	-----------

<i>valor</i>	Valor a ter seus bits deslocados para a direita.
<i>n</i>	Número de bits a ser deslocado. Deve ser do tipo <code>int</code> .

Exemplo:

```
uint valor = 12;
valor = valor >> 1;
Console.WriteLine("valor: " + valor); // 6
```

Lógicos (booleanos e bitwise)

Executam operações lógicas ou bitwise sobre seus operandos.

~

Negação (complemento de 1) bit-a-bit. Converte bits 0 em 1 e vice-versa. É definido para os tipos `int`, `uint`, `long` e `ulong`.

~*valor*

Exemplo:

```
uint valor2, valor1 = 1;
valor2 = ~valor1;
Console.WriteLine("valor2: " + valor2); // 4294967294
```

|

Definido para os tipos inteiro e `bool`. Para o tipo inteiro executa a operação OR bit-a-bit. Para o tipo `bool` executa a operação lógica OR: o resultado será `false` se ambos operandos forem `false`.

valor1 | *valor2*

Exemplo: Operandos do tipo inteiro.

```
uint valor1 = 51, valor2 = 89, valor3;
valor3 = valor1 | valor2;
Console.WriteLine("valor3: " + valor3); // 123
```