



UNIDADE VII

Linguagem de Programação C#



ALGORITMO – C Sharp (C#)

Operadores

SUMÁRIO

| | | |
|-------|--|----|
| 1 | Linguagem de programação C# | 3 |
| 2 | Tipos de Operadores | 3 |
| 2.1 | Operadores Aritméticos..... | 3 |
| 2.1.1 | Divisão Inteira | 5 |
| 2.1.2 | Concatenações de Strings | 5 |
| 2.2 | Operadores de Atribuição..... | 6 |
| 2.3 | Operadores Relacionais | 7 |
| 2.4 | Operadores Lógicos | 7 |
| 2.5 | Operador ternário “?:” | 9 |
| 2.6 | Operador “!” | 10 |
| 2.7 | Pré e Pós Incremento ou Pré e Pós Decremento | 10 |
| 2.8 | Operações com Strings | 11 |
| 2.9 | Operações com Data e Hora | 14 |
| 2.10 | Erro: Utilizar operadores incompatíveis | 14 |
| 3 | Exercícios de Fixação | 16 |
| 4 | Exercícios Complementares..... | 17 |
| 5 | Resumo da Unidade..... | 17 |

1 LINGUAGEM DE PROGRAMAÇÃO C#

Esse capítulo irá mostrar uma visão geral do C Sharp ou C#, linguagem de programação que daremos foco no nosso curso preparatório. Dando continuidade essa Unidade mostrar os principais tipos de operadores e suas funcionalidades.

2 TIPOS DE OPERADORES

Para manipular as variáveis de uma aplicação, devemos utilizar os operadores oferecidos pela linguagem de programação que estamos utilizando. A linguagem C# possui diversos operadores.

Os principais operadores dessas linguagens são:

- Aritmético: + - * / %
- Atribuição: = += -= *= /= %= ++ --
- Relacional: == != < <=

2.1 Operadores Aritméticos

Os operadores aritméticos funcionam de forma muito semelhante aos operadores da matemática.

Os operadores aritméticos são:

- Adição +
- Subtração -
- Multiplicação *
- Divisão /
- Módulo %

```
int umMaisUm = 1 + 1;
// umMaisUm = 2
int tresVezesDois = 3 * 2;
// tresVezesDois = 6
int quatroDivididoPorDois = 4 / 2;
// quatroDivididoPorDois = 2
int seisModuloCinco = 6 % 5;
// seisModuloCinco = 1
int x = 7;
x = x + 1 * 2;
// x = 9
x = x - 4;
x = x / (6 - 2 + (3 * 5) / (16 - 1));
// x = 1
```

Importante!

O módulo de um número x , na matemática, é o valor numérico de x desconsiderando o seu sinal (valor absoluto). Na matemática, expressamos o módulo da seguinte forma:

$|-2|=2$.

Em linguagens de programação, o módulo de um número é o resto da divisão desse número por outro. No exemplo acima, o resto da divisão de 6 por 5 é igual a 1. Além disso, vemos a expressão “6%5” da seguinte forma: seis módulos cinco.

Importante!

As operações aritméticas em C# obedecem as mesmas regras da matemática com relação à precedência dos operadores e parênteses. Portanto, o cálculo começa com as operações definidas nos parênteses mais internos até os mais externos. As operações de multiplicação, divisão e módulo são resolvidas antes das operações de subtração e adição.

Mais Sobre

As operações de potenciação e raiz quadrada podem ser realizadas através dos métodos `Math.Pow` e `Math.Sqrt` em C#. Veja alguns exemplos.

```
double a = Math.Pow(3, 5);
// a = 243

double b = Math.Sqrt(9);
// b = 3
```

2.1.1 Divisão Inteira

Considere uma operação de divisão entre valores inteiros. Por exemplo, uma divisão entre valores do tipo básico `int`.

```
int a = 5;
int b = 2;
Console.WriteLine(a / b);
```

Matematicamente, o resultado da operação “5 / 2” é “2.5”. Contudo, nos exemplos acima, o valor obtido na divisão “a / b” é 2. Em C#, quando ocorre uma divisão entre dois valores inteiros, a parte fracionária é descartada.

Podemos, explicitamente, converter um dos valores envolvidos na divisão ou até mesmo os dois para algum tipo numérico real. Dessa forma, a divisão não seria inteira e a parte fracionária não seria descartada. Essas conversões podem ser realizadas com operações de **casting**. Nos exemplos a seguir, o resultado das operações de divisão é 2.5.

```
int a = 5;
int b = 2;

// convertendo apenas o "a"
Console.WriteLine((double)a / b);

// convertendo apenas o "b"
Console.WriteLine((double)a / (double)b);

// convertendo apenas o "a" e o "b"
Console.WriteLine((double)a / (double)b);
```

2.1.2 Concatenações de Strings

Como vimos anteriormente, o operador `+` é utilizado para realizar soma aritmética. Mas, ele também pode ser utilizado para concatenar strings no C#. Veja alguns exemplos.

```
string s1 = " Thiago ";
string s2 = " ";
string s3 = " Sartor ";

// " Thiago Sartor"
string s4 = s1 + s2 + s3;
```

Considere os exemplos a seguir.

```
string s1 = " Idade : ";
int idade = 23;

// " Idade : 30"
string s2 = s1 + idade;
```

Observe que o operador **+** foi aplicado a valores do tipo **int** e do tipo **string**. Nesses casos, os valores do tipo **int** são, automaticamente, convertidos para **string** e a concatenação é realizada. Analogamente, essa conversão ocorrerá toda vez que o operador **+** for aplicado a valores não **string** com valores do tipo **string**.

2.2 Operadores de Atribuição

Nos capítulos anteriores, utilizamos o principal operador de atribuição, o operador **=** (igual). Os outros operadores de atribuição são:

- Simples **=**
- Incremental **+=**
- Decremental **-=**
- Multiplicativa ***=**
- Divisória **/=**
- Modular **%=**
- Incremento **++**
- Decremento **--**

```
int valor = 1; // valor = 1
valor += 2; // valor = 3
valor -= 1; // valor = 2
valor *= 6; // valor = 12
valor /= 3; // valor = 4
valor %= 3; // valor = 1
valor++; // valor = 2
valor--; // valor = 1
```

As instruções acima poderiam ser escritas de outra forma:

```
int valor = 1; // valor = 1
valor = valor + 2; // valor = 3
valor = valor - 1; // valor = 2
valor = valor * 6; // valor = 12
valor = valor / 3; // valor = 4
valor = valor % 3; // valor = 1
valor = valor + 1; // valor = 2
valor = valor - 1; // valor = 1
```

Como podemos observar, os operadores de atribuição, exceto o simples (**=**), reduzem a quantidade de código escrito. Podemos dizer que esses operadores funcionam como “atalhos” para as operações que utilizam os operadores aritméticos.

2.3 Operadores Relacionais

Muitas vezes precisamos determinar a equivalência entre duas variáveis ou a relação de grandeza (se é maior ou menor) em relação à outra variável ou valor. Nessas situações, utilizamos os operadores relacionais. As operações realizadas com os operadores relacionais devolvem valores do tipo `bool` em C#. Os operadores relacionais são:

- Igualdade `==`
- Desigualdade `!=`
- Menor `<`
- Menor ou igual `<=`
- Maior `>`
- Maior ou igual `>=`

```
int valor = 2;
bool t = false;
t = (valor == 2); // t = true
t = (valor != 2); // t = false
t = (valor < 2); // t = false
t = (valor <= 2); // t = true
t = (valor > 1); // t = true
t = (valor >= 1); // t = true
```

2.4 Operadores Lógicos

As linguagens C# permite verificar duas condições booleanas através de operadores lógicos.

Esses operadores devolvem valores do tipo `bool` em C#. A seguir descreveremos o funcionamento desses operadores.

- Os operadores “`&`”(E simples) e “`&&`”(E duplo) devolvem `true` se e somente se as duas condições forem `true`.

```
Random gerador = new Random();
double a = gerador.NextDouble();
double b = gerador.NextDouble();

Console.WriteLine(a > 0.2 & b < 0.8);
Console.WriteLine(a > 0.2 && b < 0.8);
```

A **tabela verdade** é uma forma prática de visualizar o resultado dos operadores lógicos. Veja a seguir a tabela verdade dos operadores **&** e **&&**.

| a > 0.2 | b < 0.8 | a > 0.2 & b < 0.8 | a > 0.2 && b < 0.8 |
|---------|---------|-------------------|--------------------|
| V | V | V | V |
| V | F | F | F |
| F | V | F | F |
| F | F | F | F |

- Os operadores "**|**"(OU simples) e "**||**"(OU duplo) devolvem **true** se pelo menos uma das condições for **true**.

```
Random gerador = new Random();
double a = gerador.NextDouble();
double b = gerador.NextDouble();

Console.WriteLine(a > 0.2 | b < 0.8);
Console.WriteLine(a > 0.2 || b < 0.8);
```

Também, podemos utilizar a **tabela verdade** para visualizar o resultado dos operadores **|** e **||**.

| a > 0.2 | b < 0.8 | a > 0.2 b < 0.8 | a > 0.2 b < 0.8 |
|---------|---------|-------------------|--------------------|
| V | V | V | V |
| V | F | V | V |
| F | V | V | V |
| F | F | F | F |

- O operador "**^**"(OU exclusivo) devolve **true** se apenas uma das condições for **true**.

```
Random gerador = new Random();
double a = gerador.NextDouble();
double b = gerador.NextDouble();

Console.WriteLine(a > 0.2 ^ b < 0.8);
```

Vamos visualizar resultado do operador **^** através da tabela verdade.

| a > 0.2 | b < 0.8 | a > 0.2 ^ b < 0.8 |
|---------|---------|-------------------|
| V | V | F |
| V | F | V |
| F | V | V |
| F | F | F |

Os operadores “&” e “&&” produzem o mesmo resultado lógico. Então, qual é a diferença entre eles? O operador “&” sempre avalia as duas condições. Por outro lado, o operador “&&” não avalia a segunda condição se o valor da primeira condição for falso. De fato, esse comportamento é plausível pois se o valor da primeira condição for falso o resultado lógico da operação é falso independentemente do valor da segunda condição. Dessa forma, podemos simplificar a tabela verdade do operador “&&”.

| a > 0.2 | b < 0.8 | a > 0.2 && b < 0.8 |
|---------|---------|--------------------|
| V | V | V |
| V | F | F |
| F | ? | F |

Analogamente, podemos deduzir a diferença entre os operadores “|” e “||”. As duas condições sempre são avaliadas quando utilizamos o operador “|”. Agora, quando utilizamos o operador “||” a segunda condição é avaliada somente se o valor da primeira condição for verdadeiro. Realmente, esse comportamento é aceitável pois o resultado lógico da operação é verdadeiro quando o valor da primeira condição for verdadeiro independentemente do valor da segunda condição. Dessa forma, podemos simplificar a tabela verdade do operador “||”.

| a > 0.2 | b < 0.8 | a > 0.2 b < 0.8 |
|---------|---------|--------------------|
| V | ? | V |
| F | V | V |
| F | F | F |

2.5 Operador ternário “?:”

Considere um programa que controla as notas dos alunos de uma escola. Para exemplificar, vamos gerar a nota de um aluno aleatoriamente.

```
Random gerador = new Random();
double nota = gerador.NextDouble();
```

O programa deve exibir a mensagem “aprovado” se nota de um aluno for maior ou igual a 0.5 e “reprovado” se a nota for menor do que 0.5. Esse problema pode ser resolvido com o operador ternário do C#.



Quando a condição(`nota >= 0.5`) é verdadeira, o operador ternário devolve o **primeiro resultado**(`"aprovado"`). Caso contrário, devolve o **segundo resultado**(`"reprovado"`). Podemos guardar o resultado do operador ternário em uma variável ou simplesmente exibi-lo.

```
string resultado = nota >= 0.5 ? " aprovado " : " reprovado ";
Console.WriteLine(nota >= 0.5 ? " aprovado " : " reprovado ");
```

Nos exemplos anteriores, o operador ternário foi utilizado com valores do tipo **string**. Contudo, podemos utilizá-lo com qualquer tipo de valor. Veja o exemplo a seguir.

```
int i = nota >= 0.5 ? 1 : 2;
double d = nota >= 0.5 ? 0.1 : 0.2;
```

2.6 Operador "!"

Valores booleanos podem ser invertidos com o operador de "!" (negação). Por exemplo, podemos verificar se uma variável do tipo **double** armazena um valor maior do que 0.5 de duas formas diferentes.

```
(d <= 0.5)
```

```
!(d <= 0.5)
```

2.7 Pré e Pós Incremento ou Pré e Pós Decremento

Os operadores `"++"` e `"--"` podem ser utilizados de duas formas diferentes, antes ou depois de uma variável numérica.

```
int i = 10;
i++;
i--;
```

```
int i = 10;
++i;
--i;
```

No primeiro exemplo, o operador “++” foi utilizado depois da variável *i*. Já no segundo exemplo, ele foi utilizado antes da variável *i*. A primeira forma de utilizar o operador “++” é chamada de **pós incremento**. A segunda é chamada de **pré incremento**. Analogamente, o operador “--” foi utilizado na forma de **pós decremento** no primeiro exemplo e **pré decremento** no segundo exemplo.

Mas, qual é a diferença entre **pré incremento** e **pós incremento** ou entre **pré decremento** e **pós decremento**? Vamos apresentar a diferença com alguns exemplos.

```
int i = 10;

// true
Console.WriteLine(i++ == 10);
```

Observe que o operador “++” foi utilizado nas expressões dos exemplos acima em conjunto com o operador “==”. Como dois operadores foram utilizados na mesma expressão, você pode ter dúvida em relação a ordem de execução desses operadores. O incremento com o operador “++” será realizado antes ou depois da comparação com o operador “==”?

Como o operador “++” foi utilizado na forma de **pós incremento**, a comparação ocorrerá antes do incremento. Analogamente, a comparação ocorreria antes do decremento se o operador “--” fosse utilizado na forma de **pós decremento**.

Agora, considere a utilização do operador “++” na forma de **pré incremento**.

```
int i = 10;

// false
Console.WriteLine(++i == 10);
```

Nesses últimos exemplos, a comparação com o operador “==” é realizada depois do incremento do operador “++”. Analogamente, a comparação ocorreria depois do decremento se o operador “--” fosse utilizado na forma de **pré decremento**.

2.8 Operações com Strings

Algumas operações são específicas para valores do tipo *string*. A seguir, apresentaremos algumas dessas operações.

- Descobrir a quantidade de caracteres de uma *string*.

```
string s = " Thiago Sartor ";  
  
int length = s.Length;  
  
Console.WriteLine(length);
```

- Recuperar um caractere de acordo com a sua posição na string.

```
string s = " Thiago Sartor ";  
  
char c = s[0];  
  
Console.WriteLine(c);
```

- Podemos verificar se uma determinada sequência de caracteres está contida em uma string.

```
string s = " NDD - C# e Orientação a Objetos ";  
  
bool resultado1 = s.Contains(" Java ");  
bool resultado2 = s.Contains("C#");  
  
// True  
Console.WriteLine(resultado1);  
  
// False  
Console.WriteLine(resultado2);
```

- Podemos verificar se uma string termina com uma determinada sequência de caracteres.

```
string s = " NDD - C# e Orientação a Objetos ";  
  
bool resultado1 = s.EndsWith("C#");  
bool resultado2 = s.EndsWith(" Objetos ");  
  
// False  
Console.WriteLine(resultado1);  
  
// True  
Console.WriteLine(resultado2);
```

- Podemos verificar se uma string começa com uma determinada sequência de caracteres.

```
string s = "NDD - C# e Orientação a Objetos";  
bool resultado1 = s.StartsWith("C#");  
bool resultado2 = s.StartsWith("NDD");  
// False  
System.Console.WriteLine(resultado1);  
// True  
System.Console.WriteLine(resultado2);
```

- Podemos realizar substituições em uma string.

```
string s1 = "NDD - Treinamentos";  
string s2 = s1.Replace("Treinamentos", "Cursos");  
// NDD - Cursos  
System.Console.WriteLine(s2);
```

- Podemos extrair um trecho de uma string.

```
string s1 = "Thiago Sartor";  
string s2 = s1.Substring(7);  
string s3 = s1.Substring(0, 6);  
// Sartor  
System.Console.WriteLine(s2);  
// Thiago  
System.Console.WriteLine(s3);
```

- Podemos transformar em maiúsculas todas as letras contidas em uma string.

```
string s1 = "Thiago Sartor";  
  
string s2 = s1.ToUpper();  
// THIAGO SARTOR  
System.Console.WriteLine(s2);
```

- Podemos transformar em minúsculas todas as letras contidas e numa string.

```
string s1 = "Thiago Sartor";  
  
string s2 = s1.ToLower();  
// thiago sartor  
System.Console.WriteLine(s2);
```

- Podemos eliminar os espaços em branco do começo e do término de uma string.

```
string s1 = "      Thiago Sartor";  
string s2 = s1.Trim();  
// "Thiago Sartor"  
System.Console.WriteLine(s2);
```

2.9 Operações com Data e Hora

Algumas operações são específicas para data e hora. A seguir, apresentaremos algumas dessas operações.

- Podemos modificar uma data e hora acrescentando ou subtraindo uma quantidade nos campos que definem essa data e hora.

```
System.DateTime dt = new System.DateTime(2010, 8, 27);

// Acrescentando 140 dias
dt = dt.AddDays(140);
// Subtraindo 2 anos
dt = dt.AddYears(-2);
// Acrescentando 20 segundos
dt = dt.AddSeconds(20);
```

Observe, nos exemplos acima, que 140 dias foram adicionados a data “27 de Agosto de 2010”. Automaticamente, o mês e o ano serão atualizados e a data passará a ser “14 de Janeiro de 2009”.

- Podemos comparar a ordem das datas e horas.

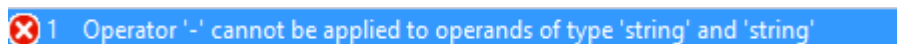
```
System.DateTime dt1 = new System.DateTime(2010, 8, 27);
System.DateTime dt2 = System.DateTime.Now;
// True
System.Console.WriteLine(dt1 < dt2);
// False
System.Console.WriteLine(dt1 > dt2);
```

2.10 Erro: Utilizar operadores incompatíveis

Um erro de compilação comum em C# ocorre quando um operador é aplicado a valores incompatíveis. Veja alguns exemplos de programas em C# com esse problema.

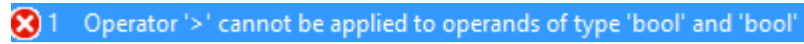
```
string s1 = "NDD";
string s2 = "Treinamentos";
System.Console.WriteLine(s1 - s2);
```

A mensagem de erro de compilação seria semelhante a apresenta abaixo.



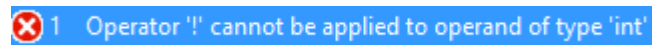
```
bool b1 = true;  
bool b2 = false;  
System.Console.WriteLine(b1 > b2);
```

A mensagem de erro de compilação seria semelhante a apresenta abaixo.

A screenshot of a C# compiler error message. It features a red square icon with a white 'X' on the left, followed by the text "1 Operator '>' cannot be applied to operands of type 'bool' and 'bool'" in white font on a blue background.

```
int i = 1;  
System.Console.WriteLine(!i);
```

A mensagem de erro de compilação seria semelhante a apresenta abaixo.

A screenshot of a C# compiler error message. It features a red square icon with a white 'X' on the left, followed by the text "1 Operator '!' cannot be applied to operand of type 'int'" in white font on a blue background.

3 EXERCÍCIOS DE FIXAÇÃO

Crie uma pasta no **GitHub** com o nome de **Operadores**.

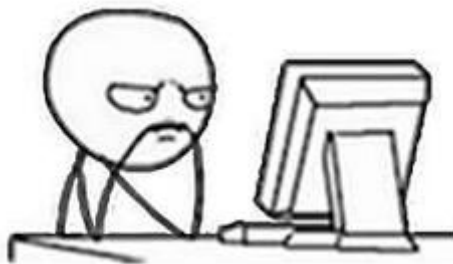
- 1) Na pasta **operadores**, implemente um programa em C# que utilize os **operadores aritméticos**.
- 2) Na pasta **operadores**, implemente um programa em C# que realize **operações de divisão inteira** e de **casting**.
- 3) Na pasta **operadores**, implemente um programa em C# que realize **operações de concatenação**.
- 4) Na pasta **operadores**, implemente um programa em C# que utilize os **operadores de atribuição**.
- 5) Na pasta **operadores**, implemente um programa em C# que utilize os **operadores relacionais**.
- 6) Na pasta **operadores**, implemente um programa em C# que utilize os **operadores lógicos**.
- 7) Na pasta **operadores**, implemente um programa em C# que utilize o **operador ternário**.
- 8) Na pasta **operadores**, implemente um programa em C# que utilize o **operador de negação**.
- 9) Na pasta **operadores**, implemente um programa em C# que utilize o operador “++” na forma de **pré e pós incremento**. Analogamente, utilize o “--” na forma de **pré e pós decremento**.
- 10) Na pasta **operadores**, implemente um programa em C# que utilize as principais **operações de strings**.

4 EXERCÍCIOS COMPLEMENTARES

Lista está no repositório **CursoNDDigital**, UNIDADE VII.

5 RESUMO DA UNIDADE

- Os **operadores** são utilizados para manipular os valores armazenados nas variáveis ou valores literais.
- As operações **aritméticas** de soma, subtração, multiplicação, divisão e resto são realizadas respectivamente através dos operadores: **+ - * / %**
- A divisão entre valores inteiros desconsidera a parte fracionária do resultado.
- O operador **+** também é utilizado para realizar a concatenação de **strings**.
- O conteúdo de uma variável pode ser modificado através dos operadores de atribuição: **= += -= *= /= %= ++ --**.
- Podemos comparar o conteúdo das variáveis ou os valores literais através dos operadores relacionais: **== != < <= > >=**.
- Operadores **relacionais** devolvem valores booleanos.
- As operações **lógicas E, OU e OU EXCLUSIVO** são realizadas através dos operadores: **& | ^ && ||**.
- O primeiro argumento do operador ternário **?:** deve ser um valor booleano.
- O operador de **negação !** inverte os valores **booleanos**.
- O operador **“++”** pode ser utilizado na forma de **pré e pós incremento**.
- O **operador “--”** pode ser utilizado na forma de **pré e pós decremento**.



Bons estudos!