

## Capítulo 4 - Operadores

Evidentemente você já resolveu alguma expressão matemática na escola. Lembra quando o professor pedia pra gente resolver aquelas expressões enormes? Cheia de números e *operadores* de somar, subtrair, multiplicar, etc. Sem falar das raízes e exponenciais. Sei que muita gente vai ter um arrepio só de lembrar disso. rs

Vamos ficar no básico ... Você sabe que na expressão  $2 + 2$  o sinal  $+$  é um operador que representa a soma. Certo? Também sabe que essa expressão resulta apenas um valor (4). Confirma? Então, você já entendeu o sentido dos *operadores*. **Relacionar valores para resultar um outro valor..**

Existem 3 tipos de *operadores*: **Aritméticos**, **Lógicos** e **Relacionais**.

Entender como funciona cada operador é MUITO importante para aprender programação. A representação simbólica de alguns operadores muda de acordo com a linguagem, mas a essência é a mesma. **Todas as linguagens de programação usam operadores.**

Vamos ver um pouquinho sobre cada tipo de operadores.

**Importante!** O conteúdo deste capítulo é um pouco denso, quero mostrar pra você como o conhecimento sobre operadores é importante. Mas você não precisa se preocupar se não entender algumas coisas que vou falar por aqui. Tome assuntos mais técnicos como curiosidade.

### Operadores Aritméticos



Esses são os mais fáceis! Aprendemos na escola fundamental. Em programação, esses operadores são tão simples quanto você aprendeu na escola. Apenas alguns que você pode não conhecer.

Além dos mais simples (soma "+", subtração "-", multiplicação "\*" e divisão "/"), dois outros operadores aritméticos

não recebem muita atenção e pode ser que você não os conheça, eles são o **div**

e o **mod**, que resultam, respectivamente, o quociente (a parte inteira do resultado da divisão) e o resto da divisão.

Peguemos por exemplo, 14 dividido por 4, vamos fazer como aprendemos na escola ...

14 dividido por 4 é igual à 3 e resta 2, certo?

Com os operadores aritméticos nós conseguimos obter o valor do quociente (4), o valor do resto (2) e o valor final da divisão (3,5).

Observe as operações abaixo:

$14 / 4 = 3,5$

$14 \text{ div } 4 = 3$

$14 \text{ mod } 4 = 2$

O operador *mod* em muitas linguagens de programação (java por exemplo) é representado pelo símbolo "%", assim:

$14 \% 4 = 2$

Uma das maiores utilizações do operador **mod** é para verificar se um número é *par* ou *ímpar*. Quando o número "**mod**" 2 resulta 0, ele é par, caso contrário, é ímpar. Veja.

$14 \text{ mod } 2 = 0 \dots \mathbf{14 \text{ é par}}$

$15 \text{ mod } 2 = 1 \dots \mathbf{15 \text{ é ímpar}}$

Um outro operador aritmético que existe em algumas linguagens de programação é o ^ e executa a operação de potência, mas geralmente essa operação é realizada através de uma função chamada **pow**, bem como a operação de radiciação (função **sqrt**). Veja um exemplo do operador ^:

$2 \wedge 5 = 32$  (dois elevado a cinco)

Operadores aritméticos de radiciação também são fornecidos por algumas linguagens de programação, mas esses são bem mais raros. O Postgres por

exemplo oferece os símbolos `//` e `///` para operações de raiz quadrada e raiz cúbica, respectivamente.

### Precedência entre os operadores

Da mesma forma que na matemática, os operadores de multiplicação e divisão têm precedência de execução em relação aos operadores de soma e subtração. Aliás se tiver parênteses na expressão estes têm precedência ainda maior.

Os operadores de mesma prioridade são interpretados da esquerda para a direita. Para exemplificar essa questão de precedência, observe a solução da expressão abaixo:

`5 + 3 * ( 3 - 1 ) - 2 ^ 5 / 4 - 1`

`5 + 3 * 2 - 2 ^ 5 / 4 - 1`

`5 + 3 * 2 - 32 / 4 - 1`

`5 + 6 - 32 / 4 - 1`

`5 + 6 - 8 - 1`

`11 - 8 - 1`

`3 - 1`

`2`

### Operadores Lógicos

Lembra dos tipos de dados que falei no capítulo anterior? Viu que dentre os tipos de dados tinha o tipo **lógico**? Então, enquanto os operadores aritméticos trabalham com números, os operadores lógicos trabalham com dados **lógicos**, ou **booleanos**.

A ideia dos operadores continua a mesma: **Relacionar valores para resultar um outro valor..** Isso significa que os operadores lógicos relacionam valores lógicos (verdadeiro/falso, 1/0, aceso/apagado, ligado/desligado, true/false,

sim/não, etc.) e o resultado de uma expressão lógica também é outro valor lógico. Simples assim!

Vamos conhecer os operadores lógicos ... Enquanto os operadores aritméticos (que você conhece muito bem) são +, -, \*, /, mod e div, os operadores lógicos são: E, OU, NÃO, NÃO-E, NÃO-OU, OU-EXCLUSIVO e NÃO-OU-EXCLUSIVO. Não é difícil, basta você acostumar (praticando) com esses operadores. Vamos ver como funciona cada um desses.

### Operador E(AND)



O Operador "E" ou "AND" resulta em um valor VERDADEIRO se os dois valores de entrada da operação forem VERDADEIROS, caso contrário o resultado é FALSO. Abaixo a **tabela-verdade** da operação E.

VALOR 1	VALOR 2	OPERAÇÃO E
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	FALSO

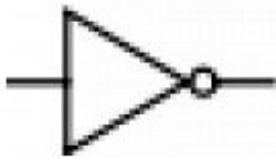
### Operador OU(OR)



O Operador "OU" ou "OR" resulta em um valor VERDADEIRO se ao menos UM dos dois valores de entrada da operação for VERDADEIRO, caso contrário o resultado é FALSO. Abaixo a **tabela-verdade** da operação OU.

VALOR 1	VALOR 2	OPERAÇÃO OU
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	FALSO

### Operador NÃO(NOT)



O Operador "NÃO" ou "NOT" é o único operador que recebe como entrada apenas um valor, e sua função é simplesmente inverter os valores. Ou seja, se o valor de entrada for VERDADEIRO, o resultado será FALSO e se o valor de entrada for FALSO, o resultado será VERDADEIRO. Abaixo a tabela-verdade da operação NÃO.

VALOR DE ENTRADA	OPERAÇÃO OU
VERDADEIRO	FALSO
FALSO	VERDADEIRO

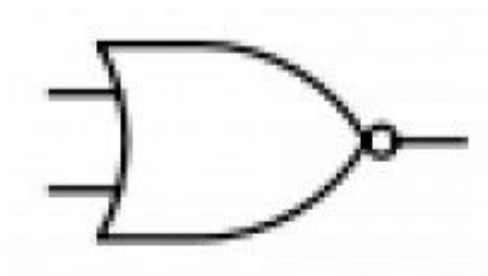
### Operador NÃO-E(NAND)



O Operador "NÃO-E" ou "NAND" é o contrário do operador E (AND), ou seja, resulta em VERDADEIRO, se ao menos um dos dois valores for FALSO, na verdade este é o operador E (AND) seguido do operador NÃO (NOT). Abaixo a **tabela-verdade** da operação NÃO-E.

VALOR 1	VALOR 2	OPERAÇÃO NÃO-E
VERDADEIRO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	VERDADEIRO

### Operador NÃO-OU(NOR)



O Operador "NÃO-OU" ou "NOR" é o contrário do operador OU (OR), ou seja, resulta em VERDADEIRO, se os dois valores forem FALSO, na verdade este é o operador OU (OR) seguido do operador NÃO (NOT). Abaixo a **tabela-verdade** da operação NÃO-OU.

VALOR 1	VALOR 2	OPERAÇÃO NÃO-OU
VERDADEIRO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	VERDADEIRO

### Operador OU-EXCLUSIVO(XOR)



O Operador "OU-EXCLUSIVO" ou "XOR" é uma variação interessante do operador OU (OR), ele resulta em VERDADEIRO se apenas um dos valores de entrada for VERDADEIRO, ou seja, apenas se os valores de entrada forem DIFERENTES. Abaixo a tabela-verdade da operação OU-EXCLUSIVO.

VALOR 1	VALOR 2	OPERAÇÃO OU-EXCLUSIVO
VERDADEIRO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO**	FALSO	FALSO

### Operador NÃO-OU-EXCLUSIVO(XNOR)



O Operador "NÃO-OU-EXCLUSIVO" ou "XNOR" é o contrário do operador OU-EXCLUSIVO (XOR), ou seja, resulta VERDADEIRO se os valores de entrada forem IGUAIS. Observe a tabela abaixo:

VALOR 1	VALOR 2	OPERAÇÃO NÃO-OU-EXCLUSIVO
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	VERDADEIRO

### Operadores lógicos em programação

Cada linguagem de programação tem uma forma de representar os operadores lógicos. A simbologia mais encontrada são:

- **AND, OR e NOT** em linguagens como: Pascal, Visual Basic e SQL.
- **&&, || e !** em linguagens como: Java e C#

No nosso caso, criando algoritmos em português, os operadores lógicos são E, OU, etc. Por exemplo.

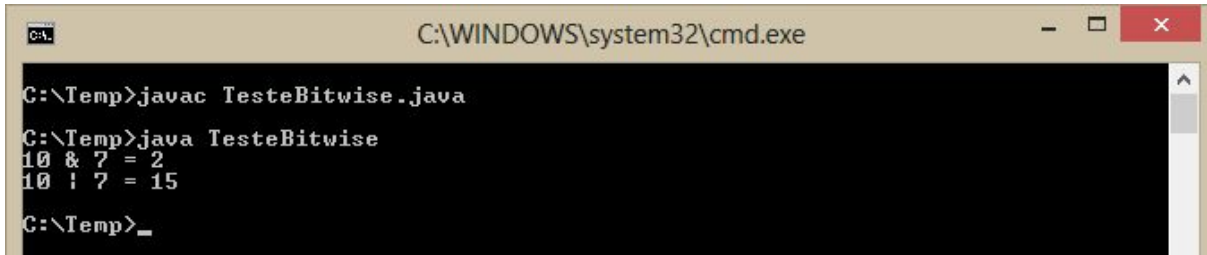
VERDADEIRO **E** FALSO = FALSO

Algumas linguagens oferecem operadores lógicos para o nível de bit (também chamado de operadores bitwise). Ou seja, podemos fazer operações lógicas com os bits de dois números. Em java, por exemplo esses operadores são & e |. Veja o código abaixo escrito em java.

```
public class TesteBitwise {  
    public static void main (String []a){  
        System.out.println("10 & 7 = " + (10 & 7));  
        System.out.println("10 | 7 = " + (10 | 7));  
    }  
}
```

```
}  
}
```

Abaixo o resultado deste programa:



```
C:\WINDOWS\system32\cmd.exe  
C:\Temp>javac TesteBitwise.java  
C:\Temp>java TesteBitwise  
10 & 7 = 2  
10 | 7 = 15  
C:\Temp>_
```

Essas operações lógicas são realizadas com os bits dos números de entrada. Assim:

Convertemos o número 10 e o número 7 para binário:

10 = 1010 em binário

7 = 0111 em binário

depois realizamos as operações lógicas com cada bit dos dois números.

Da direita para a esquerda aplicamos as operações lógicas para cada bit.

0 E 1 = 0

1 E 1 = 1

0 E 1 = 0

1 E 0 = 0



Logo,

$10 \& 7 = 0010 = 2$  em números decimais.

O mesmo para o operador OU:

$0 \text{ OU } 1 = 1$

$1 \text{ OU } 1 = 1$

$0 \text{ OU } 1 = 1$

$1 \text{ OU } 0 = 1$

Logo,

$10 | 7 = 1111 = 15$

Não é o foco aqui deste e-book, essa parte das operações em nível de bit foi só para curiosidade. Você não precisa saber fazer conversões de bases numéricas para aprender lógica de programação, mas se quiser aprender um pouco mais sobre isso, leia este post que escrevi no blog { **Dicas de Programação** }:

[As 10 conversões numéricas mais utilizadas na computação](#)

## Operadores Relacionais

Operadores relacionais são utilizados para comparar valores (de qualquer tipo), o resultado de uma expressão relacional é um valor booleano (VERDADEIRO ou FALSO). Os operadores relacionais são: **igual, diferente, maior, menor, maior ou igual, menor ou igual**.

Não é necessário explicar cada um, pois eles são auto-explicativos. Mas para quem é iniciante em desenvolvimento de softwares algumas informações podem ser importantes, principalmente pelo fato de haver diferença entre linguagens de programação.

Os operadores relacionais são diferente dependendo da linguagem de programação, mas conhecendo os símbolos mais comuns fica mais fácil

aprender. No [VisuAlg](#), os símbolos dos operadores relacionais são: =, <>, >, <, >=, <=. Vamos testar esses operadores no VisuAlg com o algoritmo abaixo que compara dois números com cada um dos operadores e mostra o resultado na tela:

```
algoritmo "TesteOperadoresRelacionais"
var
    numero1 : INTEIRO
    numero2 : INTEIRO
    resultado : LOGICO
inicio

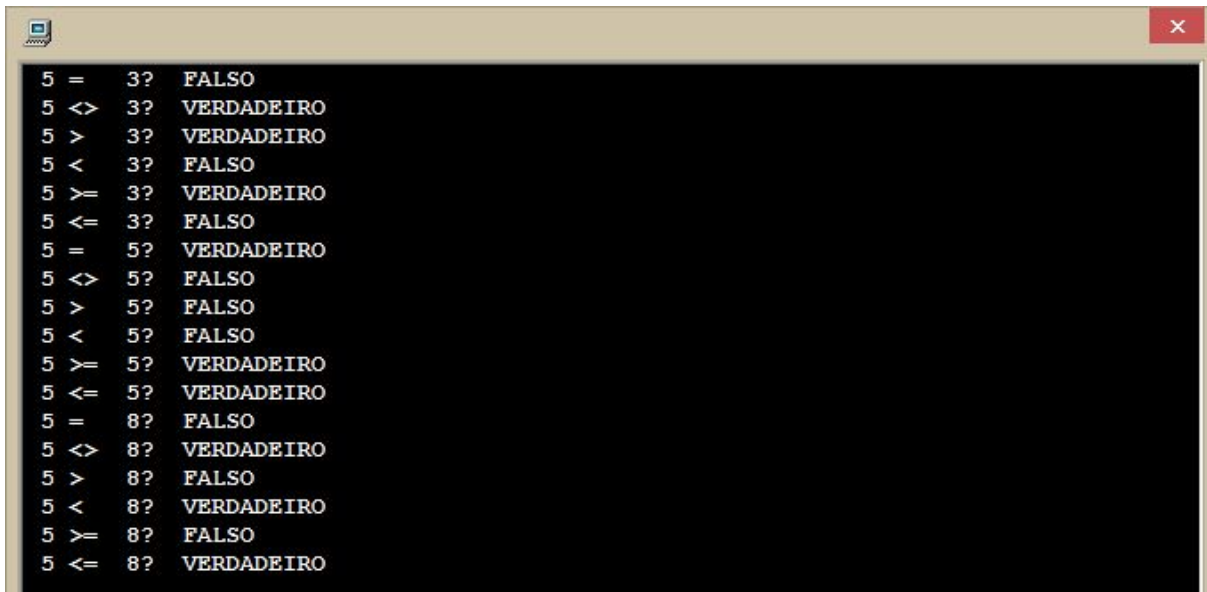
    numero1 := 5
    numero2 := 3
    resultado := numero1 = numero2
    ESCREVAL (numero1, " = ", numero2, "? ", resultado)
    resultado := numero1 <> numero2
    ESCREVAL (numero1, " <> ", numero2, "? ", resultado)
    resultado := numero1 > numero2
    ESCREVAL (numero1, " > ", numero2, "? ", resultado)
    resultado := numero1 < numero2
    ESCREVAL (numero1, " < ", numero2, "? ", resultado)
    resultado := numero1 >= numero2
    ESCREVAL (numero1, " >= ", numero2, "? ", resultado)
    resultado := numero1 <= numero2
    ESCREVAL (numero1, " <= ", numero2, "? ", resultado)

    numero1 := 5
    numero2 := 5
    resultado := numero1 = numero2
    ESCREVAL (numero1, " = ", numero2, "? ", resultado)
    resultado := numero1 <> numero2
    ESCREVAL (numero1, " <> ", numero2, "? ", resultado)
    resultado := numero1 > numero2
    ESCREVAL (numero1, " > ", numero2, "? ", resultado)
    resultado := numero1 < numero2
    ESCREVAL (numero1, " < ", numero2, "? ", resultado)
    resultado := numero1 >= numero2
    ESCREVAL (numero1, " >= ", numero2, "? ", resultado)
    resultado := numero1 <= numero2
    ESCREVAL (numero1, " <= ", numero2, "? ", resultado)
```

```
numero1 := 5
numero2 := 8
resultado := numero1 = numero2
ESCREVAL (numero1, " = ", numero2, "? ", resultado)
resultado := numero1 <> numero2
ESCREVAL (numero1, " <> ", numero2, "? ", resultado)
resultado := numero1 > numero2
ESCREVAL (numero1, " > ", numero2, "? ", resultado)
resultado := numero1 < numero2
ESCREVAL (numero1, " < ", numero2, "? ", resultado)
resultado := numero1 >= numero2
ESCREVAL (numero1, " >= ", numero2, "? ", resultado)
resultado := numero1 <= numero2
ESCREVAL (numero1, " <= ", numero2, "? ", resultado)
```

fimalgoritmo

A intenção deste algoritmo é mostrar o funcionamento dos operadores relacionais com 3 possibilidades de valores: um número menor que o outro, dois números iguais e um número maior que outro. Abaixo o resultado da execução:



5	=	3?	FALSO
5	<>	3?	VERDADEIRO
5	>	3?	VERDADEIRO
5	<	3?	FALSO
5	>=	3?	VERDADEIRO
5	<=	3?	FALSO
5	=	5?	VERDADEIRO
5	<>	5?	FALSO
5	>	5?	FALSO
5	<	5?	FALSO
5	>=	5?	VERDADEIRO
5	<=	5?	VERDADEIRO
5	=	8?	FALSO
5	<>	8?	VERDADEIRO
5	>	8?	FALSO
5	<	8?	VERDADEIRO
5	>=	8?	FALSO
5	<=	8?	VERDADEIRO

## Operadores lógicos em programação

Em todas as linguagens de programação existem símbolos para executarmos essas operações. As operações maior, menor, maior ou igual e menor ou igual na maioria das linguagens de programação são os mesmos símbolos (até hoje não encontrei uma linguagem que tenha símbolo diferente para estes operadores): > (maior), < (menor), >= (maior ou igual) e <= (menor ou igual).

Mas os vilões dos iniciantes são os símbolos para testar igualdade e diferença. Em cada linguagem é de um jeito! Em java, C, C#, javascript,... por exemplo, os símbolos de igual e diferente são: == e !=. Já em Pascal, SQL, Visual Basic, ... os símbolos de igual e diferente são: = e <>. Então fique esperto quando for aprender alguma dessas linguagens!

Em java não é possível testar Strings (textos) com o operador de igualdade (==), pois String é uma classe e não um tipo primitivo, e para testar a igualdade entre objetos deve-se utilizar o método *equals()*. Assim:

```
nome.equals("João").
```

Em algumas linguagens de programação (Python por exemplo) é possível utilizar os operadores maior e menor para verificar a precedência alfabética de um texto em relação a outro. Por exemplo: "Pedro" < "Paulo" resulta em FALSO, pois o texto "Pedro" alfabeticamente aparece depois do texto "Paulo".

Sei que neste capítulo teve bastante informação técnica e talvez você não tenha entendido algumas coisas. Não se preocupe. Se você entendeu o que são os operadores e como utilizamos eles na programação, está ótimo.

Não precisa aprofundar nos assuntos que tratamos aqui (bitwise, por exemplo), só queria te mostrar que os operadores são muito utilizados e não conhecer pelo menos o básico sobre os operadores pode comprometer o seu aprendizado de programação.

Importante lembrar que do mesmo jeito que aprendemos os operadores aritméticos nas escola, para aprender os operadores relacionais e lógicos (menos comuns) é necessário bastante prática!

Por isso, após o próximo capítulo onde você vai aprender a estrutura SE-ENTÃO-SENÃO, você poderá praticar bastante o uso desses operadores com exercícios.