

# Módulo 4 - Operadores Lógicos

## Tabela Verdade

Uma tabela verdade é uma tabela que mostra todas as possíveis combinações de valores de entrada e os resultados correspondentes de uma expressão lógica ou operação. Ela é usada para representar e analisar o comportamento de expressões lógicas em função de todas as combinações possíveis de seus operandos. Lembre-se que por **expressões lógicas**, subentende-se “verdadeiro” ou “falso”, 0 ou 1. Portanto, não existe operações lógicas com valores diferentes de 0 ou 1.

Vamos criar uma tabela verdade simples para o operador E lógico (`&&`) usando duas variáveis booleanas `A` e `B`. Aqui estão todas as combinações possíveis e os resultados:

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

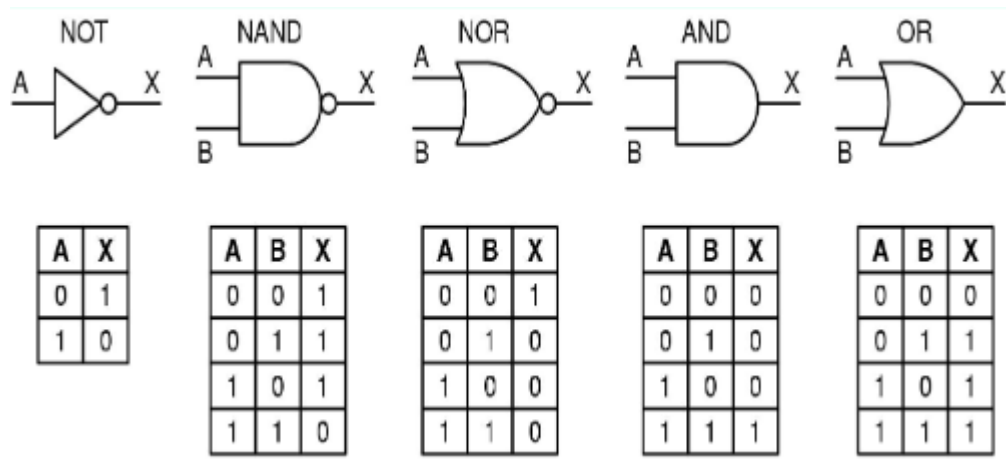
Nesta tabela:

- `A` e `B` representam as duas variáveis booleanas que estão sendo avaliadas.
- `A && B` representa o resultado da operação E lógico entre `A` e `B`.
- Os valores `0` e `1` representam falso (false) e verdadeiro (true), respectivamente.

Nas quatro combinações possíveis de `A` e `B`, a operação `A && B` resulta em verdadeiro (1) somente quando ambos `A` e `B` são verdadeiros (1).

Essa tabela verdade é útil para entender o comportamento do operador E lógico, mas tabelas verdade podem ser expandidas para expressões mais complexas com mais variáveis e operadores lógicos, permitindo a análise de condições lógicas mais elaboradas em programação ou lógica booleana.

## Todas as tabelas verdade



O `if` e o `else` da linguagem C pode realizar operações lógicas para determinar um comportamento, ela faz isso por meio dos **operadores lógicos**.

## Operadores Lógicos

Em linguagem C, os operadores lógicos são utilizados para realizar operações de lógica booleana em expressões condicionais. Eles são geralmente usados em estruturas de controle de fluxo, como `if`, `while`, `for`, entre outras, para tomar decisões com base em condições verdadeiras ou falsas. Aqui estão os operadores lógicos em C:

### 1. `&&` (E lógico):

- Sintaxe: `expressão1 && expressão2`
- Descrição: O operador `&&` retorna verdadeiro (`1`) se ambas as `expressão1` e `expressão2` forem verdadeiras (diferentes de zero).

### 2. `||` (OU lógico):

- Sintaxe: `expressão1 || expressão2`
- Descrição: O operador `||` retorna verdadeiro (`1`) se pelo menos uma das `expressão1` ou `expressão2` for verdadeira (diferente de zero).

### 3. `!` (NÃO lógico):

- Sintaxe: `!expressão`
- Descrição: O operador `!` inverte o valor de verdade de uma expressão. Se a `expressão` for verdadeira, `!expressão` será falsa, e vice-versa.

Exemplo de uso:

```
#include <stdio.h>

int main() {
    int x = 5;
    int y = 10;

    // Operador lógico E (&&)
    if (x > 0 && y > 0) {
        printf("Ambos x e y são maiores que zero.\n");
    }

    // Operador lógico OU (||)
    if (x < 0 || y < 0) {
        printf("Pelo menos um deles é menor que zero.\n");
    }

    // Operador lógico NÃO (!)
    if (!(x == 0)) {
        printf("x não é igual a zero.\n");
    }

    return 0;
}
```

Neste exemplo, os operadores lógicos `&&`, `||` e `!` são usados para tomar decisões com base nas condições especificadas.

É importante notar que as expressões em C são avaliadas como verdadeiras se seu valor for diferente de zero e falsas se seu valor for zero. Portanto, mesmo que os operadores lógicos retornem valores inteiros 1 (verdadeiro) ou 0 (falso), é comum usá-los em estruturas de controle condicionais para tomar decisões com base em condições booleanas.

Exemplo. Criando uma porta XOR:

```
#include <stdio.h>

int main(void){
    int a = 0, b = 0;

    printf("%d", (a || b) && !(a && b));
}
```

## Ordem de precedência

Depois de estudar todos estes operadores, é muito importante entender a ordem de precedência de cada um deles. Ordem de precedência entende-se pela prioridade

que o compilador dá a eles quando é feita uma operação com vários operadores ao mesmo tempo. A ordem de prioridade é a seguinte (maior para o menor):

<code>() [] -&gt; .</code>	e-d
<code>- ++ -- ! &amp; * ~ (type) sizeof</code>	d-e
<code>* / %</code>	e-d
<code>+ -</code>	e-d
<code>&lt;&lt; &gt;&gt;</code>	e-d
<code>&lt; &lt;= &gt;= &gt;</code>	e-d
<code>== !=</code>	e-d
<code>&amp;</code>	e-d
<code>^</code>	e-d
<code> </code>	e-d
<code>&amp;&amp;</code>	e-d
<code>  </code>	e-d
<code>? :</code>	d-e
<code>= op=</code>	d-e
<code>,</code>	e-d