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. Portando, 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:

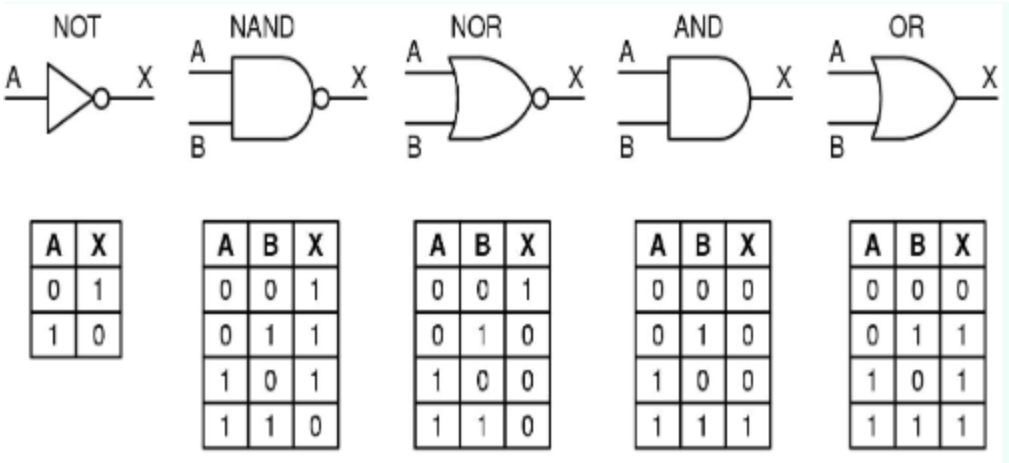
- `A` e `B` representam as duas variáveis booleanas que estão sendo avaliadas.
- `A && B` representa o resultado da operação E lógico entre `A` e `B`.
- Os valores `0` e `1` representam falso (false) e verdadeiro (true), respectivamente.

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



A	X
0	1
1	0

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

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

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

Operadores Bit a Bit

Em linguagem C, os operadores bit a bit permitem a manipulação de valores a nível de bits. Eles são usados para realizar operações bit a bit em inteiros, o que pode ser útil em várias situações, como manipulação de configurações de hardware, compactação de dados e outras operações de baixo nível. Aqui estão os principais operadores bit a bit em C:

1. `&` **(E bit a bit)**:
   - Sintaxe: `a & b`
   - Descrição: Realiza uma operação lógica E bit a bit entre os bits de `a` e `b`. O resultado é 1 se ambos os bits forem 1.
2. `|` **(OU bit a bit)**:
   - Sintaxe: `a | b`
   - Descrição: Realiza uma operação lógica OU bit a bit entre os bits de `a` e `b`. O resultado é 1 se pelo menos um dos bits for 1.
3. `^` **(OU exclusivo bit a bit)**:
   - Sintaxe: `a ^ b`
   - Descrição: Realiza uma operação lógica OU exclusivo (XOR) bit a bit entre os bits de `a` e `b`. O resultado é 1 se os bits forem diferentes.
4. `~` **(Complemento bit a bit)**:
   - Sintaxe: `~a`
   - Descrição: Inverte todos os bits de `a`, trocando 0 por 1 e vice-versa.
5. `<<` **(Deslocamento à esquerda)**:
   - Sintaxe: `a << n`
   - Descrição: Desloca os bits de `a` para a esquerda em `n` posições. Isso é equivalente a multiplicar `a` por 2 elevado a `n`.
6. `>>` **(Deslocamento à direita)**:
   - Sintaxe: `a >> n`
   - Descrição: Desloca os bits de `a` para a direita em `n` posições. Isso é equivalente a dividir `a` por 2 elevado a `n`.

Exemplo de uso:

```
#include <stdio.h>

int main() {
```

```
    unsigned int a = 12; // 1100 em binário
    unsigned int b = 25; // 11001 em binário

    unsigned int resultado;

    resultado = a & b; // Operação E bit a bit
    printf("a & b = %u\n", resultado); // Resultado: 8 (1000 em

    resultado = a | b; // Operação OU bit a bit
    printf("a | b = %u\n", resultado); // Resultado: 29 (11101 e

    resultado = a ^ b; // Operação OU exclusivo bit a bit
    printf("a ^ b = %u\n", resultado); // Resultado: 21 (10101 e

    resultado = ~a; // Complemento bit a bit
    printf("~a = %u\n", resultado); // Resultado: 4294967283

    resultado = a << 2; // Deslocamento à esquerda
    printf("a << 2 = %u\n", resultado); // Resultado: 48 (11000

    resultado = b >> 1; // Deslocamento à direita
    printf("b >> 1 = %u\n", resultado); // Resultado: 12 (1100 e

    return 0;
}
```

Lembre-se de que os operadores bit a bit geralmente são usados em contextos onde a manipulação de bits é necessária, como programação de sistemas embarcados, manipulação de protocolos de comunicação, etc.