

Condições compostas e aninhadas

Este material traz a continuação do conteúdo de condicional. Além das diversas formas de usar uma condição, veremos que ela pode ser mais elaborada e interessante. Vamos lá?

Condições compostas

As condições que vimos até agora sempre foram únicas, havia apenas uma pergunta a ser verificada, lembram disso? Entretanto, a medida que aumenta a complexidade dos problemas, temos que usar condições mais elaboradas. E como isso pode ocorrer? É o que veremos agora.

Imagine um programa que deve ler três números e determinar quantos, somente quantos são iguais.

A forma mais simples de pensar é que devo receber os três números, compará-los e ver quantos são iguais! Fácil, não é? Mas como faremos para o computador resolver essa questão?

Vamos primeiro analisar o algoritmo:

Narrativa
<ol style="list-style-type: none">1. Recebe os três números2. Verifica se os três são iguais, se sim, os três são iguais3. Verifica se os três são diferentes, se sim, nenhum número é igual4. Verifica se dois números são iguais e um diferente, se sim, dois números são iguais.

Tudo certo por enquanto? Espero que sim!

Agora, vamos traduzir isso em código, já vimos isso antes, mas é importante observar que a condição que faremos não é tão simples assim.

Na primeira condição temos que verificar se três números são iguais. Poderíamos pensar em criar a expressão (`numero1 == numero2 == numero3`). Isso resolveria nosso problema, não é? No entanto, a linguagem C não aceita esse tipo de expressão. Temos que pensar em outra solução. Teremos que criar uma expressão lógica para isso. Mas, o que é uma **expressão lógica**?

Expressão lógica é aquela que analisa duas ou mais expressões relacionais.

As expressões relacionais são as condições simples que já usamos anteriormente, na qual usamos os operadores relacionais (`==`, `!=`, `>`, `>=`, `<`, `<=`) para fazer comparações.

Então, em uma expressão lógica iremos fazer uso dos conectivos lógicos (E e OU) para unir expressões relacionais mais simples.

Vamos voltar à nossa expressão de comparação: (`numero1 == numero2 == numero3`)

Em C, essa expressão será transformada em uma expressão lógica da seguinte forma:

(`numero1 == numero2`) && (`numero 2 == numero3`)

Obs.: não precisamos fazer a verificação se numero1 é igual a numero3, pois a igualdade possui a propriedade de transitividade.

Quebramos nossa expressão anterior em duas expressões relacionais simples, e as unimos através do conectivo lógico E, que em C é representado por &&. E como esta expressão é avaliada? Como eu sei se ela corresponde ao que eu quero verificar? Vamos lá.

As expressões relacionais sempre resultam em **verdadeiro** ou **falso**. O conectivo lógico irá avaliar os valores das expressões mais simples e daí obter um resultado. A partir da combinação desses valores teremos o resultado da expressão completa.

Cada conectivo tem uma regra:

O conectivo E (&&) avaliará uma expressão como verdadeira somente quando ambas forem verdadeiras, caso contrário, será avaliada como falso.

Já o conectivo OU (||) avaliará uma expressão como falsa somente quando ambas forem falsas, caso contrário, será avaliada como verdadeiro.

Agora podemos montar o nosso código em C:


```
int main( ){
    int numero1, numero2, numero3;
    scanf ("%d %d %d", &numero1, &numero2, &numero3);
    if ((numero1 == numero2) && (numero2 == numero3))
        printf("Os tres numeros sao iguais");
    else if ((numero1 != numero2) && (numero1 != numero3) && (numero2 != numero3))
        printf ("Nao ha numeros iguais");
    else
        printf ("Dois numeros iguais e um diferente");
    return 0;
}
```

Observe que na segunda condição, temos três expressões de comparação. Então, uma expressão lógica pode ser formada por muitas expressões relacionais.

Condições aninhadas ou encadeadas

O comando de decisão (os blocos de condição) criam blocos de instruções. Esses blocos podem conter qualquer tipo de instrução, inclusive outros blocos de decisão, olha que coisa! Quando temos essa situação, um bloco de decisão dentro de outro bloco de decisão, temos o encadeamento de condições, ou as **condições aninhadas**.

Um caso particular das condições aninhadas é quando ocorre um **if** logo após o **else**. Nós vimos isso no exemplo anterior, quando logo após a verificação se os números era iguais nós verificamos se eles eram diferentes, lembram?



```
if ((numero1 == numero2) && (numero2 == numero3))
    printf("Os tres numeros sao iguais");
else if ((numero1 != numero2) && (numero1 != numero3) && (numero2 != numero3))
    printf("Nao ha numeros iguais");
```

Podemos reescrever o código inicial de uma forma diferente, usando apenas expressões simples, sem conectivos lógicos. No entanto, para isso funcionar precisamos usar condições aninhadas, ou seja, colocar uma condição dentro da outra.

Vamos ver como isso fica na prática:

```
int main( ){
    int numero1, numero2, numero3;
    scanf ("%d %d %d", &numero1, &numero2, &numero3);
    if (numero1 == numero2)
        if (numero2 == numero3)
            printf ("Os tres numeros sao iguais");
        else
            printf ("Dois numeros iguais e um diferente");
    else if (numero1 == numero3)
        printf ("Dois numeros iguais e um diferente");
    else if (numero2 == numero3)
        printf ("Dois numeros iguais e um diferente");
    else
        printf("Nao ha numeros iguais");
    return 0;
}
```

O que você achou dessa nova forma de arranjar as condições? Será que ela é mais interessante que a anterior? Olhando rapidamente ela parece mais complexa de organizar, mas será que ela pode ser mais eficiente que a anterior?

Observe também que ela necessita que o alinhamento esteja bem organizado para você não se perder na análise das expressões, é preciso que você saiba bem onde inicia e termina um determinado bloco.

Agora vá no fórum e discuta com os colegas sobre as duas formas de resolver esta questão.

Fonte: Marco Medina e Cristina Fertig, Algoritmos e programação: teoria e prática. Novatec, 2ª edição, 2005