8 Operadores Lógicos

Ronaldo F. Hashimoto e Carlos H. Morimoto

Nessa aula vamos fazer uma breve revisão sobre expressões aritméticas e relacionais, e introduzir as expressões lógicas. As expressões lógicas serão bastante utilizadas nesse curso para definir condições complexas (por exemplo, que combinam várias expressões relacionais).

Ao final dessa aula você deverá saber:

- Calcular o valor de expressões aritméticas, relacionais e lógicas.
- Utilizar operadores lógicos para a construção de condições complexas em seus programas.

8.1 Expressões Lógicas

Vimos na aula sobre comandos básicos (aula 3 que existem as expressões aritméticas e expressões relacionais. Vamos fazer uma pequena revisão.

8.1.1 Expressões Aritméticas

Expressões aritméticas são expressões matemáticas envolvendo números inteiros, variáveis inteiras, e os operadores `"+"` (soma), `"-"` (subtração), `"/"` (quociente de divisão inteira), `"%"` (resto de uma divisão inteira) e `"*"` (multiplicação).

Nessa primeira parte do curso, o resultado de uma expressão aritmética será sempre um número inteiro. Veremos mais tarde como trabalhar com expressões aritméticas com números reais.

8.1.2 Expressões Relacionais

Expressões relacionais são expressões que envolvem comparações simples envolvendo operadores relacionais `"<"` (menor), `">"` (maior), `"<="` (menor ou igual), `">="` (maior ou igual), `"!="` (diferente), `"=="` (igual). Estas expressões são normalmente usadas como <`condição`> do comando de repetição **while** e do comando de seleção **if** ou **if**−**else**

Uma comparação simples só pode ser feita entre pares de expressões aritméticas da forma:

<`expr_aritmética_01`> <`oper_relacional`> <`expr_aritmética_02`>

onde <`expr_aritmética_01`> e <`expr_aritmética_02`> são expressões aritméticas e <`oper_relacional`> é um operador relacional.

8.1.3 Expressões Lógicas

Agora vamos entrar no tópico desta aula que são as expressões lógicas que são usadas para criar condições mais complexas. Como vimos, uma expressão relacional é uma comparação que só pode ser feita entre pares de expressões aritméticas. É muito comum encontrar situações onde é necessário realizar comparações que envolvam duas expressões relacionais tais como verificar se o conteúdo de uma variável `x` é positivo (ou seja, `x`>0 - primeira expressão relacional) e ao mesmo tempo menor que 10 (ou seja, `x`<10 - segunda expressão relacional). Nesses casos vamos precisar utilizar os operadores lógicos `&&` (operador `"e"`) e || (isto mesmo,

duas barras verticais - operador "ou"). Note que o resultado de uma expressão lógica pode ser **verdadeiro** ou **falso**.

Observação:

Para verificar se o conteúdo de uma variável x é positivo (ou seja, x>0) e ao mesmo tempo menor que 10 (ou seja, x<10), **NÃO** não é correto em C escrever esta condição como $0 < x < 10$. O correto é utilizar o operador lógico && da seguinte forma (x>0) && (x<10).

Para entender porque, lembre-se que o computador "calcula" os valores de expressões, sejam elas aritméticas, relacionais ou lógicas. O valor de uma expressão é calculada segundo a precedência dos operadores. Como os operadores relacionais nesse caso tem mesma precedência, eles seriam calculados da esquerda para direita, ou seja, primeiro o computador testaria (x>0) e o resultado seria então utilizado para calcular (x<10). O problema é que o resultado de (x>0) é verdadeiro ou falso, e não faz sentido dizer se verdadeiro ou falso é menor que 10.

8.1.4 Operador Lógico &&

A tabela verdade para o operador lógico && é dada abaixo:

<expr_aritmética_01>	<expr_aritmética_02>	<expr_aritmética_01> && <expr_aritmética_02>
verdadeiro	verdadeiro	verdadeiro
verdadeiro	falso	falso
falso	verdadeiro	falso
falso	falso	falso

Observando a tabela acima, podemos concluir que o resultado do operador && é verdadeiro **APENAS** quando os dois operandos (<expr_aritmética_01> e <expr_aritmética_02>) tiverem valor verdadeiro. Se algum deles, ou ambos, tiverem valor falso, o resultado de toda expressão lógica é falso.

É verdade que basta um dos operandos (<expr_aritmética_01> ou <expr_aritmética_02>) ser falso para que toda a expressão tenha um resultado falso.

8.1.5 Operador Lógico ||

A tabela verdade para o operador lógico || é dada abaixo:

<expr_aritmética_01>	<expr_aritmética_02>	<expr_aritmética_01> || <expr_aritmética_02>
verdadeiro	verdadeiro	verdadeiro
verdadeiro	falso	verdadeiro
falso	verdadeiro	verdadeiro
falso	falso	falso

Observando a tabela acima, podemos concluir que o resultado do operador || é falso **APENAS** quando os dois operandos (<expr_aritmética_01> e <expr_aritmética_02>) tiverem valor falso. Se algum deles, ou ambos, tiverem valor verdadeiro, o resultado de toda expressão lógica é verdadeiro.

É verdade que basta um dos operandos (<expr_aritmética_01> ou <expr_aritmética_02>) ser verdadeiro para que toda a expressão tenha um resultado verdadeiro.

8.1.6 Exemplos de Expressões Lógicas

Por exemplo, assuma os seguintes valores para as variáveis inteiras x=1 e y=2. Então, veja o resultado das seguintes condições:

Expressão Lógica	Resultado
x >= y	falso
(x < y)	verdadeiro
x >= (y − 2)	verdadeiro
(x > 0) && (x < y)	verdadeiro
(y > 0)&&(x > 0)&&(x > y)	falso
(x < 0) || (y > x)	verdadeiro
(x < 0)||(y > x)&&(y < 0)	falso
para x=1 e y=2.	

8.1.7 Precedências

Os operadores lógicos também têm precedência, como descrito na tabela abaixo:

Operador Aritmético	Associatividade
*, /, %	da esquerda para a direita
+, -	da esquerda para a direita
<, >, <=, >=	da esquerda para a direita
==, !=	da esquerda para a direita
&&	da esquerda para a direita
||	da esquerda para a direita

Agora, observe a seguinte expressão lógica:

$$2 + x < y * 4 \;\&\&\; x - 3 > 6 + y / 2$$

É claro que alguém que soubesse as precedências de todos os operadores envolvidos saberia em que ordem as operações acima devem ser feitas. Agora, a legibilidade é horrível, não? Assim, uma boa dica para melhorar a legibilidade de seus programas é utilizar parênteses para agrupar as expressões aritméticas, mesmo conhecendo a precedência dos operadores, como por exemplo:

$$((2 + x) < (y * 4)) \;\&\&\; ((x - 3) > (6 + y / 2))$$

que fica muito mais fácil de se entender.

8.1.8 Exemplos em Trechos de Programa

Agora vamos ver algumas situações onde é comum utilizar expressões lógicas em programação. Considere o seguinte trecho de programa:

```
1  printf ("Entre com n > 0: ");
2  scanf ("%d", &n);
3  printf ("Entre com i > 0 e j > 0: ");
4  scanf ("%d %d", &i, &j);
5  if (n % i == 0 || n % j == 0) {
6    printf ("%d\n", n);
7  }
```

Este programa lê três números inteiros positivos `n`, `i` e `j` pelo teclado e verifica se o `n` é divisível ou por `i` ou por `j`. Em caso afirmativo, o conteúdo da variável `n` é impresso na tela. “Eta trecho de programa inútel!”. Mas serve para exemplificar um uso do operador lógico ||.

Agora, vamos para um exemplo mais “legal”. Considere o seguinte trecho de programa:

```
i=0; par=0;
while (i<10 && par == 0) {
  printf ("Entre com n > 0: ");
  scanf ("%d", &n);
  if (n % 2 == 0) {
    par = 1;
  }
  i = i + 1;
}
```

Esta repetição lê uma sequência numérica composta de no máximo 10 (dez) números inteiros positivos. Agora, se o usuário entrar somente com números ímpares, então serão lidos os 10 números ímpares e a variável `par` vai terminar com valor 0 (observe que a variável `par` foi inicializada com zero antes da repetição. Agora, se o usuário digitar algum número par, então a repetição quando retornar para verificar a condição `i<10 && par == 0` terminará (por que?) e a variável `par` terminará com valor 1 quando sair do laço.

8.1.9 Exercício

Dados `n>0` e dois números inteiros positivos `i` e `j` diferentes de 0, imprimir em ordem crescente os `n` primeiros naturais que são múltiplos de `i` ou de `j` e ou de ambos.

Exemplo: Para `n=6`, `i=2` e `j=3` a saída deverá ser : 0, 2, 3, 4, 6, 8.

Estude o trecho de código abaixo e veja se ele é uma solução para o problema acima:

```
printf ("Entre com n>0: ");
scanf ("%d", &n);
printf ("Entre com i>0: ");
scanf ("%d", &i);
printf ("Entre com j>0: ");
scanf ("%d", &j);
cont = 0; cont_num = 0;
while (cont_num < n) {
  if (cont % i == 0 || cont % j == 0) {
    printf ("%d\n", cont);
    cont_num = cont_num + 1;
  }
  cont = cont + 1;
}
```