Next Up Previous

Next: 4 Ordem sequencial de **Up:** 1 Programação Básica em **Previous:** 2 Operações Aritméticas e

Subsecções

---

3 Expressões como valores

Em C , todas as expressões são avaliadas. O resultado da avaliação é um valor e pode ser usado em quaisquer lugares.

3.1 Expressões aritméticas, relacionais e lógicas

Como você já sabe, expressões usando operadores aritméticos, relacionais e lógicos[1] são avaliados. O valor resultante é um número. Para os operadores relacionais e lógicos, este número pode ser 0 (que significa falso) ou 1 (que significa verdadeiro). Por exemplo:

`3 + 5 * 4 % (2 + 8)`	tem valor 3;
`3 < 5`	tem valor 1;
`x + 1`	tem valor igual ao valor da variável x mais um;
`(x < 1) || (x > 4)`	tem valor 1 quando o valor da variável x é fora do intervalo [1,4], e 0 quando x está dentro do intervalo.

3.2 Expressões envolvendo o operador de atribuição (=)

O formato do operador de atribuição é:

$$lvalue = expressao \quad (1)$$

Um *lvalue* (do inglês ``left-hand-side value'' - valor a esquerda) é um valor que se refere a um endereço na memória do computador. Até agora, o único ``lvalue'' válido visto no curso é o nome de uma variável. A maneira que a atribuição funciona é a seguinte: a expressão do lado direito é avaliada, e o valor é copiado para o endereço da memória associada ao ``lvalue''. O tipo do objeto do ``lvalue'' determina como o valor da *expressao* é armazenada na memória.

Expressões de atribuição, assim como expressões, têm valor. O valor de uma expressão de atribuição é dado pelo valor da expressão do lado direito do =. Por exemplo:

`x = 3`	tem valor 3;
`x = y+1`	tem o valor da expressão `y+1`.

Como consequência do fato que atribuições serem expressões que são associadas da direita para esquerda, podemos escrever sentenças como:

```
i = j = k = 0;
```

Que, usando parênteses, é equivalente a `i = (j = (k = 0))`. Ou seja, primeiro o valor 0 é atribuído a `k`, o valor de `k = 0` (que é zero) é atribuído a `j` e o valor de `j = (k = 0)` (que também é zero) é atribuído a `i`.

Uma característica muito peculiar de C é que expressões de atribuição podem ser usados em qualquer lugar que um valor pode ser usado. Porém você deve saber que usá-lo dentro de outros comandos produz um efeito colateral que é alterar o valor da variável na memória. Portanto, a execução de:

```
int quadrado, n = 2;

printf("Quadrado de %d eh menor que 50? %d \n", n, (quadrado = n * n) < 50);
```

causa não apenas que o valor `4` seja impresso, como a avaliação da expressão relacional dentro do printf() faz com que o número `4` seja copiado para o endereço de memória associado com a variável `quadrado`. Note que é necessário usar parênteses em `quadrado = n * n` já que = tem menor precedência que o operador relacional <.

Agora compare o exemplo anterior com o próximo, no qual o valor `4` é impresso, mas sem nenhum efeito colateral:

```
int quadrado, n = 2;

printf("Quadrado de %d eh menor que 50? %d \n", n, n * n < 50);
```

Note que agora não há necessidade de parênteses para a expressão `n * n` porque `*` tem maior precedência que o operador relacional <.

3.2.1 Operadores de atribuição aritmética

Como foi discutido em classe, estes comandos de atribuição funcionam de forma similar que o comando de atribuição. O lado esquerdo da expressão deve ser um *lvalue*. O valor da expressão de atribuição aritmética é igual ao valor da sentença de atribuição correspondente. Por exemplo:

`x += 3` é igual a `x = x + 3` e tem valor `x + 3`

`x *= y + 1` é igual a `x = x * (y + 1)` e tem valor `x * (y + 1)`

Notas de rodapé

... lógicos[1]

Operadores lógicos `&&` e `||` serão vistos na próxima aula.

Next Up Previous

Next: 4 Ordem sequencial de **Up:** 1 Programação Básica em **Previous:** 2 Operações Aritméticas e

Armando Luiz Nicolini Delgado

2011-02-03