Operadores em C – ref. 2

Um operador é um símbolo que indica a realização de uma operação sobre uma ou mais variáveis ou valores. Há uma grande diversidade de operadores em C, sendo os principais apresentados nesta seção.

Operadores aritméticos

Os principais operadores aritméticos são descritos abaixo. O "módulo" é o resto da divisão inteira. Considere que A=10 e B=20.

Operador	Descrição	Exemplo	Resultado
+	soma	A + B	30
-	subtração	A - B	-10
*	multiplicação	A * B	200
/	divisão inteira	B / A	2
%	módulo	B % A	0
++	incremento	A++	11
--	decremento	A--	9

Os operadores de incremento e decremento têm dois sabores, pré- e pós-fixados. No caso pré-fixado, (++a), o incremento ocorre antes que o valor seja usado; no caso pós-fixado, (a++), o incremento ocorre depois que o valor for utilizado. Suponha que n=5, então

```
x = n++;    // atribui e depois incrementa: x=5, n=6
y = ++n;    // incrementa e depois atribui: y=7, n=7
```

faz com que 5 seja atribuído a x, e faz com que 7 seja atribuído a y. Após a execução dos dois comandos, n=7.

Operadores relacionais

Conside que A=10 e B=20.

Operador	Descrição	Exemplo	Resultado
==	igual	A == B	FALSE
!=	diferente	A != B	TRUE
>	maior do que	A > B	FALSE
<	menor do que	A < B	TRUE
>=	maior ou igual	A >= B	FALSE
<=	menor ou igual	A <= B	TRUE

Operadores lógicos

Considere que A=TRUE e B=FALSE.

Operador	Descrição	Exemplo	Resultado
&&	AND lógico	A && B	FALSE
||	OR lógico	A || B	TRUE
!	NOT lógico	!A	FALSE

Operadores de atribuição

Considere que A=10 e B=20.

Operador	Descrição	Exemplo	Equivale
=	atribuição simples de valor a variável	A = B + 17	
+=	atribuição com soma	A += 3	A = A + 3
-=	atribuição com subtração	A -= B - 7	A = A - (B - 7)
*=	atribuição com produto	C *= A	C = C * A
/=	atribuição com divisão	C /= A	C = C / A
%=	atribuição com módulo	C %= A	C = C % A
<<=	atribuição com deslocamento à esquerda	C <<= 2	C = C << 2
>>=	atribuição com deslocamento à direita	C >>= 2	C = C >> 2
&=	atribuição com AND binário	C &= 2	C = C & 2
|=	atribuição com OR binário	C |= 2	C = C | 2
^=	atribuição com XOR binário	C ^= 2	C = C ^ 2

Muito cuidado para não confundir a atribuição "=" (igual) com o teste de igualdade "==" (igual igual).

Por exemplo, a expressão ''if (x = 3)'' será sempre TRUE, independentemente do valor de ''x''. Por mais esquisito que pareça, C permite atribuições no meio de expressões.

Em C, o "valor" de uma atribuição é o valor atribuído.

As atribuições-com-operação ('+=', '*=') são, frequentemente, mais naturais do que a atribuição simples; pensamos "incremente x de 5" (x += 5;) quando queremos fazer "x=x+5",
mas *não* pensamos "adicione cinco a x e atribua o resultado para x".

Operadores bit-a-bit

Considere duas variáveis inteiras de 8 bits sem sinal:

- A = 0001 0001 (17)
- B = 0110 0011 (99)

Operador	Descrição	Exemplo	Resultado
&	AND bit-a-bit	A & B	0000 0001 (1)
|	OR bit-a-bit	A | B	0111 0011 (115)

Operador	Descrição	Exemplo	Resultado
^	XOR bit-a-bit	A ^ B	0111 0010 (114)
~	complemento de 1	~A	1110 1110 (238)
<<	desloca à esquerda N bits	A << 2	0100 0100 (68)
>>	desloca à direita N bits	A >> 2	0000 0100 (4)

Operadores diversos

Operador	Descrição	Exemplo
sizeof()	tamanho da variável em bytes	sizeof(A)
&	endereço de variável	&A
*	posição apontada em memória	*x
? :	condicional ternária: *if X then Y else Z*	X ? Y : Z

Ordem de precedência

A ordem de precedência dos operadores vistos acima define a sequência em que os operadores serão aplicados. A ordem de precedência pode ter forte influência sobre o resultado da avaliação de expressões complexas.

A tabela a seguir indica a ordem de precedência dos principais operadores em C, da maior para a menor. Todos os operadores associam da esquerda para a direita, exceto os indicados na descrição.

Ordem	Operadores	Descrição
1	() [] -> .	chamadas de função acesso a vetores e estruturas
2	! ~ - + * & sizeof casting ++ --	operadores unários **(dir. p/ esq)**
3	* / %	multiplicação, divisão, módulo
4	+ -	adição, subtração
5	<< >>	deslocamentos
6	< > <= >=	relacionais
7	== !=	relacionais
8	&	AND bit-a-bit (0110 & 0110)
9	^	XOR bit-a-bit (0110 ^ 0110)
10	|	OR bit-a-bit (0110 | 0110)
11	&&	AND lógico (TRUE && FALSE)
12	||	OR lógico (TRUE || FALSE)
13	? :	expressão condicional ternária **(dir. p/ esq)**
14	= += -= *= /= %= &= |= ^= <<= >>=	atribuições **(dir. p/ esq)**
15	,	operador vírgula (*e ainda*)

A formatação livre de C nos permite abusar da legibilidade de muitas formas. Por exemplo, o que significa o comando (válido) abaixo?

```
x = a+++b;
```

poderia ser

```
x = a + ++b;
```

ou

```
x = a++ + b;
```

As regras de precedência resolvem este caso:

```
x = a + (++b);  // ++ tem maior precedência do que +
```

Exercícios

1. (Kelley&Pohl) Considerando a=1, b=2, c=3, d=4, e=5 compute as expressões abaixo.

```
a * b / c
a * b % c + 1
++ a * b - c --
7 - - b * ++ d
a / b / c
7 + c * -- d / e
2 * a % - b + c + 1
39 / - ++ e - + 29 % c
7 - + ++ a % (3+b)
```

2. (Kelley&Pohl) Considerando i=1, j=2, k=3, m=4, compute as expressões abaixo.

```
i += j + k
j *= k = m + 5
i += j += k += 1 + 2
```

3. (K&R) A função bitcount conta o número de bits em 1 no seu argumento. Corte o programa abaixo, compile-o e altere os valores atribuídos a a,b,c e verifique os resultados.

```
#include <stdio.h>              // inclusão de definições

int bitcount(unsigned int);     // declaração da função


int main (void)                 // função principal
{
   int a, b, c, n;              // declaração das variáveis

   a = 10;                      // valores para os lados dos triângulos
   b = 100;
   c = 1000;

   n = bitcount(a);
   printf ("a: %6d 0x%04x \t bits: %d\n", a, a, n);  // imprime resultado na tela

   n = bitcount(b);
   printf ("b: %6d 0x%04x \t bits: %d\n", b, b, n);  // imprime resultado na tela
```

```
   n = bitcount(c);
   printf ("c: %6d 0x%04x \t bits: %d\n", c, c, n);  // imprime resultado na tela

  return (0);                     // retorno (fim) da função
}

int bitcount(unsigned int x)
{
    int b;

    for (b = 0; x != 0; x >>= 1) // atribuição com deslocamento
       if (x & 1)
          b++;
    return b;
}
```

A *string* nos printf() significa:

"a: ", depois um decimal (%d) alinhado à direita em seis 'casas' (%6d), depois um hexadecimal (%x) alinhado à direita em quatro 'casas' (%4x) e completado à esquerda com zeros (%04x), depois um TAB ('\t'), " bits: ", depois um inteiro (%d), seguido de um NEWLINE ('\n'). A variável **a** é exibida em dois formatos, decimal e hexadecimal para benefício dos humanos embora ela seja armazenada na memória em binário.

4) (K&R) Na representação em complemento de dois, "x &= (x-1)" apaga o bit menos significativo de x (bit mais à direita). Explique o porquê. Use esta otimização para escrever uma versão mais rápida de bitcount.<>