

C

Operadores e Expressões

Adriano Cruz
`adriano@nce.ufrj.br`

Instituto de Matemática
Departamento de Ciência da Computação
UFRJ

15 de agosto de 2013

Section Summary

- 1 Introdução
- 2 Operador de Atribuição
- 3 Operadores Aritméticos
- 4 Operadores Relacionais e Lógicos
- 5 Operadores Lógicos
 - E lógico
 - OU lógico
 - Não lógico
- 6 Operadores com Bits
- 7 Operadores de Atribuição Composta
- 8 Operador vírgula
- 9 Operador **sizeof()**
- 10 Conversão de Tipos

- ① Adriano Cruz. *Curso de Linguagem C*, Disponível em <http://equipe.nce.ufrj.br/adriano>
- ② Ulysses de Oliveira. *Programando em C*, Editora Ciência Moderna.

Operadores e Expressões

O objetivo deste capítulo é apresentar os operadores existentes na linguagem C e a forma correta de construir expressões que envolvam estes operadores, constantes e variáveis.

Section Summary

- 1 Introdução
- 2 Operador de Atribuição
- 3 Operadores Aritméticos
- 4 Operadores Relacionais e Lógicos
- 5 Operadores Lógicos
 - E lógico
 - OU lógico
 - Não lógico
- 6 Operadores com Bits
- 7 Operadores de Atribuição Composta
- 8 Operador vírgula
- 9 Operador **sizeof()**
- 10 Conversão de Tipos

- Este é o operador usado para transferir o resultado de uma expressão para uma variável.
- Em C este operador é o sinal de igual (=).
- Este sinal não está representando que o resultado da expressão do lado direito é igual ao resultado do lado esquerdo e sim uma atribuição.
- Observe que o comando de atribuição termina em ponto e vírgula.
- `soma = a + b; pi = 3.1415;`

Multiplas Atribuições

- É possível fazer-se várias atribuições em uma única linha, como no exemplo a seguir:
- `a = b = c = 1.0;`
- As três variáveis recebem o mesmo valor.
- As atribuições são feitas na seguinte ordem:
 - 1 `c = 1.0;` c recebe o valor 1.0.
 - 2 b recebe o resultado da expressão à sua direita, que é o valor atribuído à c, ou seja 1.0.
 - 3 a recebe o resultado da expressão à sua direita, que é o valor atribuído à b, ou seja 1.0.

Section Summary

- 1 Introdução
- 2 Operador de Atribuição
- 3 Operadores Aritméticos
- 4 Operadores Relacionais e Lógicos
- 5 Operadores Lógicos
 - E lógico
 - OU lógico
 - Não lógico
- 6 Operadores com Bits
- 7 Operadores de Atribuição Composta
- 8 Operador vírgula
- 9 Operador **sizeof()**
- 10 Conversão de Tipos

Operadores Aritméticos

Operador	Descrição	Prioridade
+	Mais unário	0
-	Menos unário	0
++	Incremento	1
--	Decremento	1
*	Multiplicação	2
/	Divisão	2
%	Resto da divisão	2
+	Soma	3
-	Subtração	3

Regras de Precedência

- Outro ponto importante são as regras de precedência que determinam que operação deve ser executada primeiro.
- Na tabela os operadores estão listados em ordem decrescente de prioridade.
- Para os operadores aritméticos a operação de mais alta precedência é o $-$ unário, vindo em seguida $++$, $--$ com a mesma prioridade.
- Os operadores de multiplicação ($*$), divisão ($/$) e módulo ($\%$) tem a mesma prioridade.
- O operador menos unário multiplica seu operador por -1 .
- Quando duas operações de mesmo nível de prioridade têm de ser avaliadas, a operação mais à esquerda será avaliada primeiro.

Parênteses

- Parênteses têm um papel importante nas expressões e permitem que a ordem das operações seja alterada.
- Expressões entre parênteses são calculadas em primeiro lugar, portanto eles conferem o maior grau de prioridade as expressões que eles envolvem.
- Podemos ter pares de parênteses envolvendo outros pares. Dizemos que os parênteses estão aninhados.
- Neste caso as expressões dentro dos parênteses mais internos são avaliadas primeiro.

```
int i = 3; int j = 6; int k = 1;  
int r = i / (j / k);
```

- Um ponto importante que deve ser sempre levado em consideração quando uma expressão for calculada são os tipos das variáveis, porque eles alteram radicalmente os resultados das expressões.
- Por exemplo, a divisão entre operandos do tipo inteiro tem como resultado um valor inteiro.
- Portanto, se o resultado possuir uma parte fracionária ela será truncada.
- Não é possível aplicar a operação de módulo a operandos do tipo float e double.
- Por exemplo a operação $1 / 3$ em C fornece como resultado o valor 0, enquanto que $1 \% 3$ é igual a 1.

Exemplos

- $a + \frac{b}{b+c} \implies a + b/(b+c)$
- $b^2 + c^2 \implies b*b + c*c$
- $\frac{x}{a+\frac{b}{c}} \implies x/(a+b/c)$

Section Summary

- 1 Introdução
- 2 Operador de Atribuição
- 3 Operadores Aritméticos
- 4 Operadores Relacionais e Lógicos
- 5 Operadores Lógicos
 - E lógico
 - OU lógico
 - Não lógico
- 6 Operadores com Bits
- 7 Operadores de Atribuição Composta
- 8 Operador vírgula
- 9 Operador **sizeof()**
- 10 Conversão de Tipos

Operadores Relacionais

Operador	Descrição	Prioridade
>=	Maior ou igual a	0
>	Maior que	0
<=	Menor ou igual a	0
<	Menor que	0
==	Igual a	1
!=	Diferente de	1

- Os operadores $>$, $>=$, $<$ e $<=$ têm a mesma precedência e estão acima de $==$ e $!=$.
- Estes operadores têm precedência menor que os aritméticos.
- Portanto expressões como $(i < limite - 1)$ e $i < (limite - 1)$ têm o mesmo significado.

Section Summary

- 1 Introdução
- 2 Operador de Atribuição
- 3 Operadores Aritméticos
- 4 Operadores Relacionais e Lógicos
- 5 Operadores Lógicos
 - E lógico
 - OU lógico
 - Não lógico
- 6 Operadores com Bits
- 7 Operadores de Atribuição Composta
- 8 Operador vírgula
- 9 Operador **sizeof()**
- 10 Conversão de Tipos

Operadores Lógicos

- Os operadores lógicos definem as maneiras como as relações acima podem ser conectadas.
- Por exemplo podemos querer testar se ao mesmo tempo uma nota é maior ou igual a 5.0 e a taxa de presença é maior que 75%.
- Para simplificar a apresentação destes operadores serão usadas variáveis para substituir as relações.
- Neste caso a expressão acima seria representada como p e q , onde p está representando nota maior ou igual a 5.0 e q taxa de presença maior que 75%.

Operadores Lógicos

- Estas expressões podem ter dois resultados verdadeiro e falso.
- Observar que, assim como em operações aritméticas, podemos ter combinações de mais de duas relações em uma única expressão.
- Por exemplo, podemos ter a seguinte combinação: ano maior que 2000 e mês menor que 6 e dia maior que 15.
- Nas linguagens de programação os valores verdadeiro e falso podem ser representados de diversas maneiras.
- Em C o valor falso é representado por 0 e verdadeiro por qualquer valor diferente de 0.

Operador &&

p	q	p && q
0	0	0
0	1	0
1	0	0
1	1	1

Operador &&

- O símbolo usado para representar o operador E lógico é &&
- O resultado da expressão é verdadeiro se e somente se todas as variáveis forem iguais a verdadeiro.



```
int i = 3, j = -5;  
float z = 3.0;  
int resultado;
```

```
resultado = (10 > 5) && ( i > j) && (z != 0);  
printf("O resultado e vale %d.", resultado);
```

- O resultado deste trecho é a impressão de um valor diferente de 0, ou o valor correspondente a verdadeiro.
- Isto porque (10 é maior que 5) E (i é maior que j) E (z é diferente de 0).

Operador ||

p	q	p q
0	0	0
0	1	1
1	0	1
1	1	1

Operador ||

- O símbolo usado para representar o operador OU lógico é ||.
- Para que o resultado da expressão seja verdade basta que uma das variáveis seja verdade.

- ```
float x = 3.0;
int n = 55, i = 0;
int resultado;

resultado = (i != 0) || (x == 0) || (n < 100);
printf("O resultado é %d", resultado);
```

- O resultado deste trecho é a impressão do valor 1.
- Apesar de i não ser diferente de 0 e x não ser diferente de zero, temos que n é menor que 100.
- Como basta um dos testes ser verdade para o resultado ser verdade será impresso um valor diferente de 0.

# Operador !

- O símbolo usado para representar o operador NÃO lógico é !.
- Este operador é unário e quando aplicado à uma variável ele troca seu valor.



```
int dia = 25, ano = 1959;
int resultado;

resultado = ! ((dia < 30) && (ano > 1950))
printf ("O resultado vale %d.", resultado);
```

- Este trecho de programa imprime 0 (falso), porque dia é menor que 30 E ano é maior que 1950.
- Portanto, o resultado do parênteses vale verdadeiro. No entanto, o operador ! nega este valor que vira 0.



# Operador !

| p | !p |
|---|----|
| 0 | 1  |
| 1 | 0  |

Tabela: Operador Lógico NÃO.

# Precedência Relacionais e Lógicos

| Operador     | Prioridade |
|--------------|------------|
| !            | 0          |
| >, >=, <, <= | 1          |
| ==, !=       | 2          |
| &&           | 3          |
|              | 4          |

# Section Summary

- 1 Introdução
- 2 Operador de Atribuição
- 3 Operadores Aritméticos
- 4 Operadores Relacionais e Lógicos
- 5 Operadores Lógicos
  - E lógico
  - OU lógico
  - Não lógico
- 6 Operadores com Bits
- 7 Operadores de Atribuição Composta
- 8 Operador vírgula
- 9 Operador **sizeof()**
- 10 Conversão de Tipos

# Operadores com bits

- Para operações com bits, a linguagem C dispõe de alguns operadores que podem ser usados nos tipos `char`, `int`, `long` e `long long`.
- Não podem ser usados em `float`, `double`, `long double` e `void`.
- A diferença entre estes operadores e os lógicos é que estes operam em pares de bits enquanto que os operadores lógicos anteriores consideram a palavra toda.
- Por exemplo, para um valor `int` ser falso é necessário que todos os 32 bits sejam iguais a zero.

# Operadores com bits

| Operador | Descrição             | Prioridade |
|----------|-----------------------|------------|
| »        | Desloca para direita  | 0          |
| «        | Desloca para esquerda | 0          |
| ~        | Não                   | 1          |
| &        | E                     | 2          |
| ^        | Ou exclusivo          | 3          |
|          | Ou                    | 4          |

# Operadores com bits

| p | q | $p \wedge q$ | $p \& q$ | $p \mid q$ | $\sim p$ |
|---|---|--------------|----------|------------|----------|
| 0 | 0 | 0            | 0        | 0          | 1        |
| 0 | 1 | 0            | 0        | 1          | 1        |
| 1 | 0 | 0            | 0        | 1          | 0        |
| 1 | 1 | 1            | 1        | 1          | 0        |

# Operadores de deslocamento

Os operandos de deslocamento têm os seguintes modos de operação:

- operando  $\gg$  vezes: o operando é deslocado vezes bits para a direita.
- operando  $\ll$  vezes: o operando é deslocado vezes bits para a esquerda.

# Operadores de deslocamento

- Nos deslocamentos à direita em variáveis `unsigned` e nos deslocamentos à esquerda, os bits que entram são zeros;
- Nos deslocamentos à direita em variáveis `signed`, os bits que entram correspondem ao sinal do número (1= sinal negativo, 0 = sinal positivo).
- Um deslocamento para a direita é equivalente a uma divisão por 2. Deslocamento para a esquerda é equivalente a uma multiplicação por 2. Assim  $a = a * 2$ ; e  $a = a \ll 1$ ; são equivalentes.



# Exemplo

```
#include <stdio.h>
int main (void)
{
 unsigned int c = 7;
 int d = -7;

 c = c<<1; printf("%3d = %08X\n", c, c);
 c = c>>1; printf("%3d = %08X\n", c, c);

 d = d<<1; printf("%3d = %08X\n", d, d);
 d = d>>1; printf("%3d = %08X\n", d, d);

 return 0;
}
```

# Resultados

|     |   |          |
|-----|---|----------|
| 14  | = | 0000000E |
| 7   | = | 00000007 |
| -14 | = | FFFFFFF2 |
| -7  | = | FFFFFFF9 |

- Os resultados mostram que o número 7 após o primeiro deslocamento de 1 bit para a esquerda ficou igual a 14, portanto um 0 entrou no número.
- Quando o número foi deslocado para direita 1 bit, ele retornou ao valor original.
- Observe que quando o número -14 foi deslocado para a direita entrou um bit 1, que é igual ao sinal negativo.

# Section Summary

- 1 Introdução
- 2 Operador de Atribuição
- 3 Operadores Aritméticos
- 4 Operadores Relacionais e Lógicos
- 5 Operadores Lógicos
  - E lógico
  - OU lógico
  - Não lógico
- 6 Operadores com Bits
- 7 Operadores de Atribuição Composta
- 8 Operador vírgula
- 9 Operador **sizeof()**
- 10 Conversão de Tipos

# Atribuição Composta

- Em C qualquer expressão da forma:  
`variavel = variavel operador expressao`
- pode ser escrita como:  
`variavel operador= expressao`
- Por exemplo:  
`ano = ano + 10;`
- é equivalente a  
`ano += 10;`

# Outros Exemplos

- `raiz = raiz * 4;`
- `raiz *= 4;`
- `soma = soma / ( a + b );`
- `soma /= (a + b);`
- `a = a » 1;`
- `a »= 1;`
- `i = i % 2;`
- `i %= 2;`

# Section Summary

- 1 Introdução
- 2 Operador de Atribuição
- 3 Operadores Aritméticos
- 4 Operadores Relacionais e Lógicos
- 5 Operadores Lógicos
  - E lógico
  - OU lógico
  - Não lógico
- 6 Operadores com Bits
- 7 Operadores de Atribuição Composta
- 8 Operador vírgula
- 9 Operador `sizeof()`
- 10 Conversão de Tipos

# Operador vírgula

- O operador vírgula (,) é usado para separar duas ou mais expressões que são escritas onde somente uma é esperada.
- Quando o conjunto de expressões tem de ser reduzido a somente um valor, somente a expressão mais à direita é considerada.
- Por exemplo, considere o seguinte trecho de código:

```
y = (x=5, x+2);
```

- A expressão começa a ser avaliada da esquerda para a direita.
- Portanto, primeiro seria atribuído o valor 5 a variável x.
- Em seguida atribui x+2 para a variável y.
- Ao final a variável x contém o valor 5 e y o valor 7.

# Section Summary

- 1 Introdução
- 2 Operador de Atribuição
- 3 Operadores Aritméticos
- 4 Operadores Relacionais e Lógicos
- 5 Operadores Lógicos
  - E lógico
  - OU lógico
  - Não lógico
- 6 Operadores com Bits
- 7 Operadores de Atribuição Composta
- 8 Operador vírgula
- 9 Operador **sizeof()**
- 10 Conversão de Tipos



# Operador `sizeof()`

- O operador `sizeof()` é um operador unário que retorna o tamanho em bytes da expressão ou tipo fornecido entre parênteses.
- Por exemplo, suponha que o tipo `float` tenha quatro bytes então o operador `sizeof(float)` retorna o valor 4.
- Para se calcular o tamanho de bytes de uma expressão não é necessário o uso de parênteses.

# Exemplo

```
#define DIM 10
#include <stdio.h>
int main()
{
 int i=0;
 float f=3.0;
 char c='a';
 int v[DIM];

 printf("Tamanho de int %d\n", sizeof i);
 printf("Tamanho do float %d\n", sizeof f);
 printf("Tamanho do double %d\n", sizeof (↵
 double));
 printf("Tamanho do char %d\n", sizeof c);
 printf("Tamanho do vetor de %d inteiros %d\n",
 DIM, sizeof(v)).
```

# Resultados

```
Tamanho em bytes de alguns tipos
Tamanho de int 4
Tamanho do float 4
Tamanho do double 8
Tamanho do char 1
Tamanho do vetor de 10 inteiros 40
```

# Section Summary

- 1 Introdução
- 2 Operador de Atribuição
- 3 Operadores Aritméticos
- 4 Operadores Relacionais e Lógicos
- 5 Operadores Lógicos
  - E lógico
  - OU lógico
  - Não lógico
- 6 Operadores com Bits
- 7 Operadores de Atribuição Composta
- 8 Operador vírgula
- 9 Operador `sizeof()`
- 10 Conversão de Tipos

- Quando operandos de tipos diferentes aparecem em expressões são convertidos para um tipo comum, que permita o cálculo da expressão da forma mais eficiente.
- Por exemplo, uma operação que envolva um tipo `int` e um `float`, o valor `int` é convertido para `float`.
- Por exemplo, em uma divisão de inteiros o resultado é do tipo inteiro.
- A expressão  $1/3*3$  tem como resultado o valor inteiro 0.

- Operandos do tipo `char` e `int` podem ser livremente misturados em expressões aritméticas.
- Os tipos `char` são convertidos para `int`.
- Por exemplo, a conversão de uma letra maiúscula para minúscula pode ser facilmente implementada com o comando:  
`l = l - 'A' + 'a';`

# Regras de Conversão

- `char` é convertido para `int`;
- `float` é convertido para `double`.
- Se algum dos operandos é `double` o outro é convertido para `double` e o resultado é `double`.
- Caso contrário, se algum dos operandos é `long`, o outro é convertido para `long` e o resultado é `long`.
- Caso contrário, se algum dos operandos é `unsigned`, o outro é convertido para `unsigned` e o resultado é deste tipo.
- Caso contrário os operandos são `int` e o resultado é `int`.
- Note que todos os `floats` em uma expressão são convertidos para `double` e a expressão é avaliada em `double`.
- O resultado de uma expressão é convertido para o tipo da variável onde o resultado será armazenado.
- Um resultado `float` ao ser carregado em uma variável do tipo `int` causa o truncamento da parte fracionária, porventura existente.

# Regras de Conversão

- A conversão de inteiro para caractere é bem comportada, mas o contrário nem sempre ocorre convenientemente.
- A linguagem não especifica se o tipo `char` é um tipo com sinal ou não.
- Quando um caractere é armazenado em uma variável do tipo inteiro podem ocorrer problemas com caracteres que têm o bit mais à esquerda igual a 1.
- Isto porque algumas arquiteturas podem estender este bit e outras não.



# Regras de Precedência

| Pri | Operador            | Descrição                                 |
|-----|---------------------|-------------------------------------------|
| 0   | () [] -> .          | Agrupamento; acesso vetor; acesso membro  |
| 1   | ! ~ ++ - + -        | Unárias lógicas, aritméticas              |
| 1   | * & (tipo) sizeof() | ponteiros, conformação de tipo; tamanho   |
| 2   | * / %               | Multiplicação, divisão e módulo           |
| 3   | + -                 | soma e subtração                          |
| 4   | » «                 | Deslocamento de bits à direita e esquerda |
| 5   | < <= >= >           | Operadores relacionais                    |
| 6   | == !=               | Igualdade e diferença                     |
| 7   | &                   | E bit a bit                               |
| 8   | ^                   | Ou exclusivo bit a bit                    |
| 9   |                     | Ou bit a bit                              |
| 10  | &&                  | E                                         |
| 11  |                     | Ou                                        |
| 12  | ? () : ()           | Ternário                                  |
| 13  | = += -= *= /= %=    | Atribuições                               |
| 13  | >= <= &=  =         | Atribuições                               |
| 14  | ,                   | Separador de expressões                   |

The End