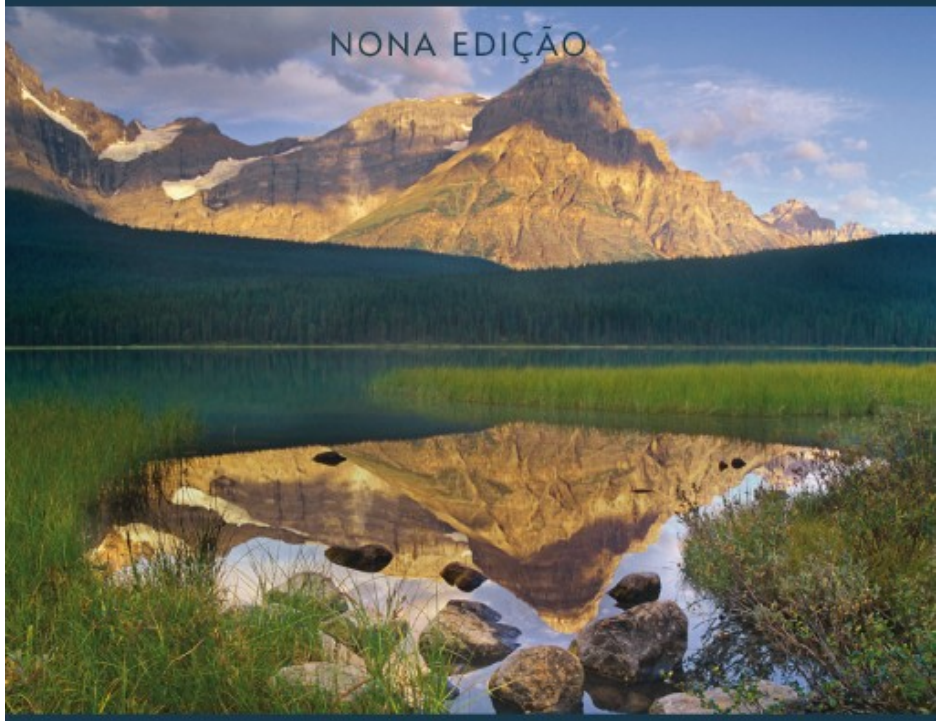
CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO

NONA EDIÇÃO

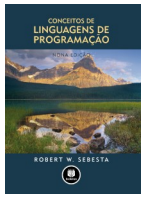


ROBERT W. SEBESTA

bookman

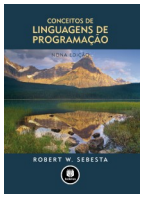
Capítulo 7

Expressões e Sentenças de Atribuição



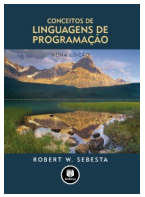
Tópicos do Capítulo 7

- Introdução
- Expressões aritméticas
- Operadores sobrecarregados
- Conversões de tipos
- Expressões relacionais e booleanas
- Avaliação em curto-circuito
- Sentenças de atribuição
- Atribuição de modo misto



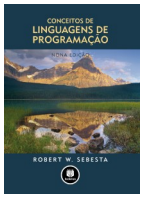
Introdução

- Expressões são os meios fundamentais de especificar computações em uma linguagem de programação
- Para entender a avaliação de expressões, é necessário estar familiarizado com as ordens de avaliação de operadores e operandos
- A essência das linguagens imperativas é o papel dominante das sentenças de atribuição



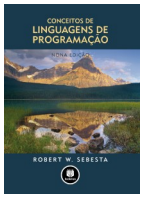
Expressões aritméticas

- Expressões aritméticas
- Avaliação aritmética foi uma das motivações para o desenvolvimento das primeiras linguagens de programação
- Expressões aritméticas consistem em operadores, operandos, parênteses e chamadas a funções



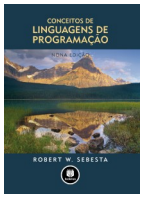
Expressões aritméticas: questões de projeto

- Quais são as regras de precedência de operadores?
- Quais são as regras de associatividade de operadores?
- Qual é a ordem de avaliação dos operandos?
- Quais são os efeitos colaterais na avaliação de operandos?
- A linguagem permite a sobrecarga de operadores definida pelo usuário?
- Que tipo de mistura de tipos é permitida nas expressões?



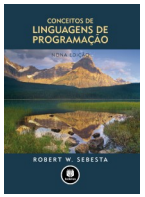
Expressões aritméticas: operadores

- Um operador unário tem um operando
- Um operador binário tem dois operandos
- Um operador ternário tem três operandos



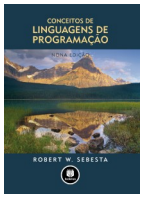
Expressões aritmeticas: regras de precedência de operadores

- As *regras de precedência de operadores* para avaliação de expressões definem a ordem pela qual os operadores de diferentes níveis de precedência são avaliados
- Níveis de precedência mais usadas
  - parênteses
  - operadores unários
  - ** (se a linguagem suporta)
  - *, /
  - +, -



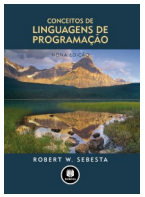
Expressões aritméticas: regras de associatividade de operadores

- As *regras de associatividade de operadores* para a avaliação de expressões definem a ordem em que ocorrências adjacentes de operadores com o mesmo nível de precedência são avaliados
- Regras de associatividade mais usadas
  - Da esquerda para a direita, exceto **, que é da direita para a esquerda
  - Operadores unários às vezes associam da direita para a esquerda (por exemplo, em FORTRAN)
- APL é diferente; todos os operadores têm o mesmo nível de precedência e associam da direita para a esquerda
- Regras de precedência e associatividade podem ser alteradas com parênteses



Expressões em Ruby

- Operadores aritméticos, relacionais e de atribuição, como índices de matrizes, deslocamentos e operadores lógicos bit a bit, são implementados como métodos
  - Um resultado disso é que esses operadores podem ser sobrescritos por programas de aplicação



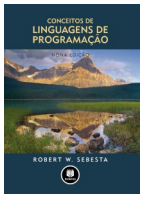
Expressões aritméticas: expressões condicionais

- Expressões condicionais
  - Linguagens baseadas em C (como C e C++)
  - Um exemplo:

```
average = (count == 0)? 0 : sum / count
```

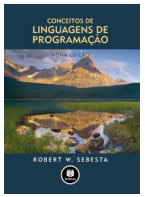
  - Avalia como se fosse escrita como

```
if (count == 0)
   average = 0
else
   average = sum /count
```



Expressões aritméticas: ordem de avaliação de operandos

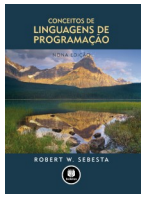
- *Ordem de avaliação de operandos*
  1. Variáveis: obtêm o valor a partir da memória
  2. Constantes: algumas vezes avaliadas da mesma maneira; em outros casos, é parte da instrução de linguagem de máquina
  3. Expressões entre parênteses: avaliam todos os operadores que ela contêm antes de seu valor ser usado como operando
  4. O caso mais interessante surge quando a avaliação de um operando tem efeitos colaterais



Expressões aritméticas: efeitos colaterais

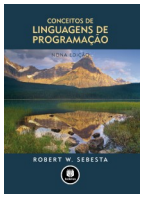
- *Efeitos colaterais funcionais:* quando a função modifica um de seus parâmetros ou uma variável global
- Problema com efeitos colaterais funcionais:
  - Quando uma função referenciada em uma expressão altera outro operando da expressão; por exemplo, para uma mudança de parâmetro:

```
a = 10;
b = a + fun(&a);
```



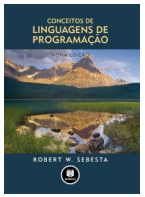
Efeitos colaterais

- Duas possíveis soluções para o problema
  1. Escrever a definição de linguagem para proibir efeitos colaterais funcionais
     - Não para parâmetros de duas direções
     - Não para variáveis globais
     - **Vantagem:** funciona!
     - **Desvantagem:** limitado a parâmetros de uma direção e falta de referências globais
  2. Dizer na definição da linguagem que os operandos em expressões devem ser avaliados em uma certa ordem
     - **Desvantagem:** limita algumas otimizações do compilador
     - Java garante que os operandos sejam avaliados da esquerda para a direita



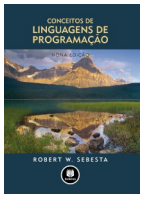
Operadores sobrecarregados

- Usar um operador para mais de um propósito é chamado de *sobrecarga de operadores*
- Alguns são comuns (por exemplo, `+` para `int` e `float`)
- Alguns são problema em potencial (por exemplo, `*` em C e C++)
  - Perda de detecção de erro do compilador (omissão de um operando deve ser um erro detectável)
  - Alguma perda da legibilidade



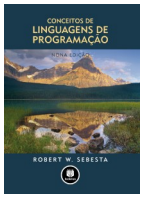
Operadores sobrecarregados

- C++ e C# permitem operadores sobrecarregados definidos pelo usuário
- Problemas em potencial:
  - Usuários podem definir operadores sem sentido
  - Facilidade de leitura pode ser prejudicada, mesmo quando os operadores fazem sentido



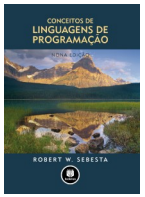
Conversões de tipos

- Uma *conversão de estreitamento* converte um valor para um tipo que não pode armazenar aproximações equivalentes a todos os valores do tipo original. Por exemplo, `float` para `int`
- Uma *conversão de alargamento* converte um valor para um tipo que pode incluir ao menos aproximações de todos os valores do tipo original. Por exemplo, `int` para `float`



Conversões de tipo: modo misto

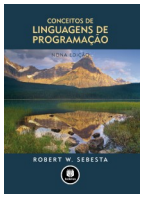
- Uma *expressão de modo misto* tem operandos de tipos diferentes
- Uma *coerção* é um tipo implícito de conversão
- Desvantagem de coerções:
  - Eles diminuem a capacidade de detecção de erros do compilador
- Na maioria das linguagens, todos os tipos numéricos têm coerção nas expressões, usando conversões de alargamento
- Em Ada, praticamente não há coerções nas expressões



Conversão do tipo explícita

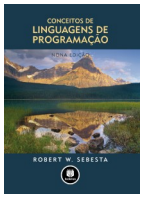
- Chamadas de *cast* em linguagens baseada em C
- Exemplos
  - `C: (int)angle`
  - `Ada: Float (Sum)`

Em Ada, os cast têm a sintaxe de chamadas a funções



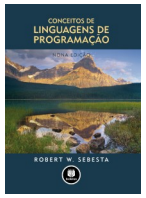
Erros em expressões

- Causas
  - Limitações inerentes da aritmética por exemplo, divisão por zero
  - Limitações da aritmética computacional por exemplo, transbordamento
- Muitas vezes ignorado pelo tempo de execução do sistema



Expressões relacionais e booleanas

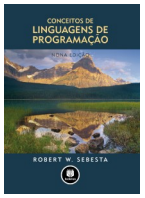
- Expressões relacionais
  - Usam operadores relacionais e operandos de vários tipos
  - Avaliar para alguma representação booleana
  - Símbolos de operação variam um pouco entre as linguagens (!=, /=, ~=, .NE., <>, #)
- JavaScript e PHP têm dois operadores relacionais adicionais, === e !==
  - Similares a seus operadores relativos, == e !=, mas previnem que seus operandos sofram coerção



Expressões relacionais e booleanas

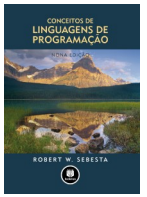
- Expressões booleanas
  - Operandos são booleanos e o resultado é booleano
  - Exemplos de operadores

FORTRAN 77	FORTRAN 90	C	Ada
.AND.	and	&&	and
.OR.	or	||	or
.NOT.	not	!	not
			xor



Expressões relacionais e booleanas: sem tipo booleano em C

- C89 não tem um tipo booleano - usa o tipo `int` com 0 para falso e todos diferentes de zero como verdadeiro
- Uma característica estranha de expressões C: **`a < b < c`** é uma expressão legal, mas o resultado pode não ser esperado:
  - Operador da esquerda é avaliado, produzindo 0 ou 1
  - O resultado da avaliação é então comparado com o terceiro operando (no exemplo, **c**)



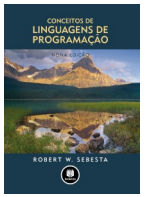
Avaliação em curto-circuito

- Uma expressão na qual o resultado é determinado sem avaliar todos os operandos e/ou operadores
- Exemplo: **`(13*a) * (b/13–1)`**

  Se a é zero, não há necessidade de avaliar **`(b/13-1)`**
- Problema com avaliação que não é em curto-circuito

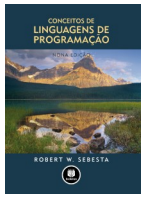
```
index = 1;
while (index <= length) && (LIST[index] != value)
      index++;
```

  – Quando **`index=length`**, **`LIST [index]`** causa um erro de indexação (assumindo que **`LIST`** tem elementos **`length -1`**)



Avaliação em curto-circuito

- C, C++ e Java: usa avaliação em curto-circuito para operadores booleanos usuais (**`&&`** e **`||`**), mas também fornece operadores booleanos bit a bit que não são em curto-circuito (**`&`** e **`|`**)
- Ada: programador pode especificar ambos (curto-circuito é especificado com **`and then`** e **`or else`**)
- Avaliação em curto-circuito expõe o problema potencial de efeitos colaterais em expressões
  por exemplo, **`(a > b) || (b++ / 3)`**



Sentenças de atribuição

- A sintaxe geral

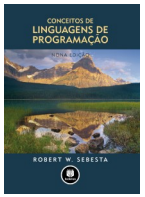
  `<target_var> <assign_operator> <expression>`

- O operador de atribuição

  `=` FORTRAN, BASIC e linguagens baseadas em C

  `:=` ALGOLs, Pascal, Ada

- = pode ser ruim quando está sobrecarregado para o operador relacional de igualdade (por isso que as linguagens baseadas em C usam == como operador relacional)



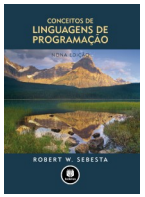
Sentenças de atribuição: alvos condicionais

- Alvos condicionais (Perl)

```
($flag ? $total : $subtotal) = 0
```

que é equivalente a

```
if ($flag){
  $total = 0
} else {
  $subtotal = 0
}
```



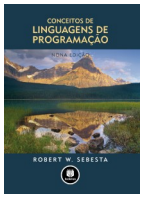
Sentenças de atribuição: operadores compostos

- É um método de atalho para especificar uma forma de atribuição comumente necessária
- Introduzida em ALGOL; adotada por C
- Exemplo

```
a = a + b
```

é escrito como

```
a += b
```



Sentenças de atribuição: operadores de atribuição unários

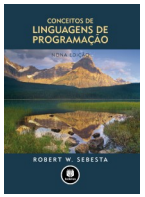
- Operadores de atribuição unários em linguagens baseadas em C combinam operações de incremento e decremento com atribuição
- Exemplos

  `sum = ++count` (`count` incrementado, adicionado a `sum`)

  `sum = count++` (`count` incrementado, adicionado a `sum`)

  `count++` (`count` incrementado)

  `-count++` (`count` incrementado então negado)

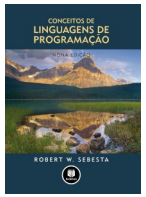


Atribuição como uma expressão

- Em C, C++ e Java, a sentença de atribuição produz um resultado e pode ser usada como operandos
- Um exemplo:

```
while ((ch = getchar())!= EOF){...}
```

`ch = getchar()` é realizado; o resultado (atribuído a `ch`) é usado como valor condicional para a sentença `while`

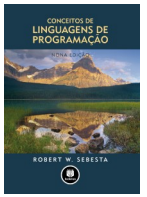


Atribuições de listas

- Perl e Ruby suportam atribuições de lista

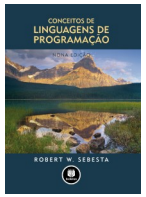
  por exemplo,

```
($first, $second, $third) = (20, 30, 40);
```



Atribuição de modo misto

- Sentenças de atribuição também podem ser de modo misto
- Em Fortran, C e C++, qualquer valor de tipo numérico pode ser atribuído a uma variável de tipo numérico
- Em Java, apenas se a coerção requerida é de alargamento
- Ada não permite atribuição de modo misto



Resumo

- Expressões
- Precedência e associatividade de operador
- Sobrecarga de operador
- Expressões de modo misto
- Várias formas de atribuição