

# Expressões e sentença de atribuição

---

Marco A L Barbosa  
malbarbo.pro.br

Departamento de Informática  
Universidade Estadual de Maringá



Este trabalho está licenciado com uma Licença Creative Commons - Atribuição-CompartilhaIgual 4.0 Internacional.  
<http://github.com/malbarbo/na-lp-copl>

# Conteúdo

Introdução

Expressões aritméticas

Sobrecarga de operadores

Conversões de tipos

Expressões relacionais e booleanas

Avaliação em curto-circuito

Sentenças de atribuição

Atribuição em modo misto

Referências

# Introdução

- Expressões
  - É a maneira fundamental de especificar computações em linguagens de programação
  - É essencial que o programador entenda a sintaxe e a semântica das expressões
  - Entender a semântica requer conhecer a ordem de avaliação dos operadores e operandos, regras de conversão de tipos, etc

- Atribuição
  - O propósito da atribuição é alterar o valor de uma variável
  - Tem papel dominante nas linguagens de programação imperativas

# **Expressões aritméticas**

## Expressões aritméticas

- A maioria das características das expressões aritméticas em linguagens de programação são baseadas em convenções da matemática
- Uma expressão aritmética consiste em operadores, operandos, parênteses e chamadas de funções
- Um operador pode ser unário (1 operando), binário (2 operandos), ternário (3 operandos), etc
- Existem 3 maneiras comuns de especificar a aplicação de um operador: prefixa, infix e posfixa

## Questões de projeto

- Quais são as regras de precedência dos operadores?
- Quais são as regras de associatividade dos operadores?
- Qual é a ordem de avaliação dos operandos?
- Existem restrições no efeito colateral da avaliação de um operando?
- A linguagem permite sobrecarga de operador definida pelo programador?
- Que mistura de tipos é permitida nas expressões?



# Ordem de avaliação dos operadores

- A ordem de avaliação dos operadores depende das
  - Regras de precedência
  - Regras de associatividade

- Qual o resultado da avaliação da expressão  $3 + 4 * 5$ ?  
Depende das regras de precedência
- As regras de **precedência de operadores** para avaliação de expressões definem a ordem que os operadores com diferentes níveis de precedência devem ser avaliados

---

## Linguagens baseadas em C

---

Mais alta	++ e -- posfixados ++ e -- prefixados e + e - unários *, /, %
Mais baixa	+ e - binários

---

- Em Lisp, a questão de precedência não se aplica
  - $(+ 3 (* 4 5))$
  - $(* (+ 3 4) 5)$
- Pyret não tem precedência
  - $(3 + 4) * 5$
  - $3 + (4 * 5)$
  - $1 + 2 + 4$

- Considere a expressão  $a - b + c - d$ . Se a adição e subtração tiverem a mesma precedência, qual operação será realizada primeiro? A ordem de execução das operações pode alterar o resultado da expressão?
  - Depende das regras de associatividade
  - Sim! Podem haver problemas com números reais e com inteiros muito grandes

- As regras de **associatividade de operadores** para avaliação de expressões definem a ordem que os operadores adjacentes com o mesmo nível de precedência devem ser avaliados
- Comumente a associatividade é da esquerda para direita, exceto a exponenciação que é da direita para esquerda

# Associatividade

Linguagem	Associatividade
Linguagens baseadas em C	Esquerda: *, /, %, + e - binários Direita: ++, --, + e - unários

- Em APL todos os operadores tem a mesma precedência e a associatividade é da direita para esquerda



- O programador pode alterar as regras de precedência e associatividade usando parênteses

# Expressões em Ruby

- Ruby é uma linguagem orientada a objetos pura
- Todos os dados são objetos, inclusive os literais
- Quase todos os operadores são chamadas de métodos
- Por exemplo, a expressão `a + b` especifica a chamada do método `+` do objeto referenciado por `a`
- Estes operadores podem ser sobrescritos como qualquer outro método

## Expressões condicionais

- Em algumas linguagens, o `if` é uma expressão, o que permite (entre outras coisas) atribuir o resultado da execução de um `if` a uma variável

```
x = if cond then exp_a else exp_b
// que é equivalente (e mais conveniente) que
if cond then
    x = exp_a
else
    x = exp_b
```

# Expressões condicionais

- As linguagens baseadas em C têm o operador ternário ?:

```
x = cond? exp_a : exp_b
```

- Python oferece um variação do **if**

```
x = exp_a if cond else exp_b
```

# Ordem de avaliação dos operandos

- Forma de avaliação dos operandos
  - Variável: o valor é lido da memória
  - Constante: o valor é lido da memória ou faz parte da instrução da máquina
  - Expressão parentizada: todos os operandos e operadores devem ser avaliados
  - Função: deve ser executada

- Um **efeito colateral** de uma função ocorre quando a função altera um de seus parâmetros ou uma variável global
- Expressões em linguagens puramente funcionais e na matemática não produzem efeitos colaterais

## Ordem de avaliação dos operandos

- Qual é o valor de a após a execução da função main?

```
int a = 5;
int fun() {
    a = 17;
    return 3;
}
void main() {
    a = a + fun();
}
```

- Depende da ordem da avaliação dos operandos a e fun(), pois o operando fun() gera o efeito colateral de alterar o valor de a

# Ordem de avaliação dos operandos

- Como resolver o problema da ordem de avaliação de operadores e efeitos colaterais?
  - Não permitir efeitos colaterais
    - Vantagens: permite algumas otimizações pelo compilador
    - Desvantagens: difícil implementação, limitações para o programador
  - Definir uma ordem de avaliação dos operandos (Java)
    - Vantagens: não limita o programador
    - Desvantagens: limita algumas otimizações pelo compilador



- Um programa tem a propriedade de **transparência referencial** se quaisquer duas expressões no programa que tenham o mesmo valor puderem ser substituídas uma pela outra em qualquer lugar do programa, sem afetar as ações do programa
- Transparência referencial de função: o valor de uma função depende apenas dos seus parâmetros

## Ordem de avaliação dos operandos

- O conceito de transparência referencial está relacionado com o efeitos colaterais de funções. Exemplo

```
result1 = (fun(a) + b) / (fun(a) - c);
```

```
temp = fun(a);
```

```
result2 = (temp + b) / (temp - c)
```

- Os valores `result1` e `result2` devem ser iguais se a função `fun` não tem efeito colateral
- Quais as vantagens da transparência referencial?

# Sobrecarga de operadores

# Sobrecarga de operadores

- **Sobrecarga de operador** é a utilização de um operador para mais que um propósito
- Geralmente aceitável, desde que não comprometa a leitura e a confiabilidade
  - Aceitável: +, usado para somar inteiros e floats
  - Nem tanto (segundo o Sebesta): &, usado para endereço e e-bit-a-bit

## Sobrecarga de operadores

- Algumas linguagens permitem sobrecarregar os operadores padrão (C++, C#, Ruby, Python, etc).
- Ajuda na legibilidade quando usado de forma coerente

`A * B + C * D`

vs

`MatrixAdd(MatrixMul(A, B), MatrixMul(C,D))`

## **Conversões de tipos**

# Conversões de tipos

- As conversões podem ser
  - De **estreitamento**, quando um objeto é convertido para um tipo que não pode incluir todos os valores do tipo original. Ex: `float` para `int`
  - De **ampliação**, quando um objeto é convertido para um tipo que contém pelo menos aproximações para os valores do tipo original. Ex: `int` para `float`
  - Em geral, as conversões de ampliação são seguras
- Uma **expressão em modo misto** é aquela que tem operandos de tipos diferentes

- Conversão **implícita** (coerção)
  - Feita implicitamente pelo compilador / interpretador
  - Utilizado nas expressões em modo misto
  - Na maioria das linguagens, os tipos numéricos podem sofrer conversões implícitas de ampliação
  - Em Ada, não existem coerções
  - Vantagem: aumenta a flexibilidade
  - Desvantagem: diminui a utilidade da checagem de tipo



- Conversão **explícita**
  - Feita explicitamente pelo programador
  - O compilador pode gerar um alerta sobre conversões de estreitamento
  - Exemplos
    - C: `(float) angle`
    - Ada: `Float(angle)`

## **Expressões relacionais e booleanas**

# Expressões relacionais e booleanas

- Um **operador relacional** é um operador que compara os valores de dois operandos
- Uma **expressão relacional** tem dois operandos e um operador relacional
- Exemplos de operadores relacionais
  - igualdade: =, ==, ===
  - desigualdade: !=, <>, /=, ~=
- Os operadores relacionais em geral têm prioridade menor que os operadores aritméticos

a + 1 > 2 \* b

# Expressões relacionais e booleanas

- Uma **expressão booleana** consiste de variáveis ou constantes booleanas, expressões relacionais e operadores booleanos
- Os operadores booleanos comuns são: **and**, **or** e **not**
- Na maioria das linguagens o **and** tem prioridade sobre o **or**
- Em Ada os operadores **and** e **or** tem a mesma prioridade e não são associativos
- Os operadores booleanos em geral têm prioridade menor que os operadores relacionais

$x > y$  **and**  $x \neq 4$

- C/C++
  - Qual o significado da expressão  $x > y > z$ ?
- Python
  - A expressão  $x > y > z$  tem o significado esperado, isto é  $x > y$  **and**  $y > z$

## **Avaliação em curto-circuito**

- Uma **avaliação em curto-circuito** de um expressão é aquela que o resultado é determinado sem avaliar todos os operandos e/ou operadores
- Por exemplo, na expressão  $(13 * a) * (b / 13 - 1)$ , se  $a = 0$  não é necessário avaliar  $(b / 13 - 1)$  para determinar o resultado da expressão, que é zero

## Avaliação em curto-circuito

- Em geral, a avaliação em curto circuito é utilizada em expressões booleanas

- Possibilidade de construções do tipo

```
while ((index < listlen) && (list[index] != key))  
    index++;
```

- Problemas com efeitos colaterais

```
(a > b) || ((b++) / 3)
```

- C/C++, Java usam os operadores em curto circuito && e ||, e operadores que não são curto circuito & e |
- Ada usa **and then** e **or else** para especificar que a expressão deve ser avaliada em curto circuito



## **Sentenças de atribuição**

# Sentenças de atribuição

- A forma geral para atribuição é
  - alvo símbolo-de-atribuição expressão
- O símbolo de atribuição
  - = Fortran e as linguagens baseadas em C
  - := Algol, Pascal e Ada

## Sentenças de atribuição

- Alvos condicionais (Perl)

```
$(flag ? $count1 : $count2) = 0;
```

```
# é equivalente a
```

```
if ($flag) {  
    $count1 = 0;  
} else {  
    $count2 = 0;  
}
```

# Sentenças de atribuição

- Atribuição composta

`a += 1`

que é equivalente a

`a = a + 1`

- Atribuição unária

`count++`

- Atribuição como expressão
  - Em C/C++, Java, uma atribuição gera um valor que pode ser usado como operando

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

# Sentenças de atribuição

- Listas de atribuições
  - Suportada por Perl, Ruby, Python
  - Python

```
x, y = 1, 6
```

```
p = [7, 8]
```

```
a, b = p # extrai os valores da lista p
```

```
a, b = b, a # trocas os valores
```

**Atribuição em modo misto**

## Atribuição em modo misto

- Os tipos das expressões tem que ser o mesmo da variável sendo atribuída ou uma coerção pode ser feita?
- Diferente de C++, Java e C# permiti apenas casos de coerções de ampliação



## Referências

- Robert Sebesta, Concepts of programming languages, 9ª edição. Capítulo 7.