Expressões e Comandos de Atribuição

Paradigmas de Programação – BCC/UFRPE
Lucas Albertins – lucas.albertins@deinfo.ufrpe.br

Agenda

- Introdução
- Expressões Aritméticas
- Efeitos Colaterais
- Transparência Referencial
- Sobrecarga de Operadores
- Expressões Relacionais
- Avaliação em Curto-Circuito
- Comandos de Atribuição

Introdução

- Expressões são a forma fundamental para especificar computações em linguagens de programação
- Para entender avaliações de expressões é necessário ser familiarizado com as ordens de avaliação dos operandos e operadores
- Comandos de atribuição são essenciais em linguagens imperativas

Expressões Aritméticas

- Avaliações aritméticas foi uma das motivações para o desenvolvimento das primeiras linguagens de programação
- Expressões aritméticas consistem de operadores, operandos, parênteses e chamadas a funções
- Operadores
  - unário, binário, ternário
  - Infix, prefix ou postfix

Expressões Aritméticas

- Decisões de projeto:
  - Regras de precedência dos operadores?
  - Regras de associatividade dos operadores?
  - Ordem de avaliação de operandos?
  - Efeitos colaterais na avaliação de operandos?
  - Sobrecarga de operadores?
  - Quais tipos são permitidos nas expressões?

Expressões Aritméticas – Ordem de Avaliação

- **Precedência** (hierarquia de prioridades de avaliação)
  - a + b * c
  - Em geral: parênteses, op. Unários, potência, * / % e + -
- **Associatividade**
  - Ordem em que operadores adjacentes com mesmo nível de precedência são avaliados
  - Mais comum: da esquerda para a direita
    - A - B + C
    - a = 4; b = 2; c = 10;
- Tanto as regras de **precedência** quanto **associatividade** podem ser sobrescritas com parênteses

Precedência

	Ruby	Linguagens Baseadas em C
Maior	**	Postfix ++, --
	Unário +, -	Prefix ++, --, unário +, -
	*, /, %	*, /, %
Menor	Binário +, -	Binário +, -

Associatividade

Linguagem	Linguagens Baseadas em C
Ruby	Esquerda: *, /, +, -
	Direita: **
C	Esquerda: *, /, %, binária +, binária +
	Direita: ++, --, unária -, unária +
Ada	Esquerda: Todas exceto **
	Não-associativa **

Expressões em Ruby e LISP

- Ruby
  - Todos os valores de dados são objetos
  - Operadores aritméticos são métodos
  - Ex: a+b
    - Chamada ao método + do objeto a, passando b como parâmetro
  - Útil para aplicar sobrecarga em operadores utilizados para tipos definidos pelos usuários
- LISP
  - Todos as operações lógicas e aritméticas são chamadas a subprogramas
  - a + b * c => (+ a (* b c))

Expressões condicionais

- Comandos if-then-else podem realizar uma atribuição
- Expressao1 ? Expressao2 : Expressao3 (operador ternário)
- Ex em C:
  - `media = (cont==0)? 0: soma/cont;`
- Também presentes em Perl, JavaScript e Ruby

Efeitos colaterais

- Efeitos colaterais de funções: função altera o parâmetro or variável global

```
/* assuma que fun retorna 10 e alterar o
parametro para 20 */
a = 10;
b = a + fun(&a);
```

Soluções?

Transparência Referencial

- Duas expressões no programa que têm o mesmo valor podem ser substituídas uma pela outra em qualquer lugar do programa sem afetá-lo

```
result1 = (fun(a) + b) / (fun(a) - c);
temp = fun(a);
result2 = (temp + b) / (temp - c);
```

- `result1 = result2` Transparência Referencial OK
- Facilita o entendimento
- Programas em linguagens funcionais puras são referencialmente transparentes

Sobrecarga de Operadores

- Aceitável se não afetar a legibilidade nem a confiabilidade
- Ex:
  - + para inteiros e reais
  - & para AND e para endereço de varíavel
- C++ e C# permitem sobrecarga de operadores
- Potenciais problemas:
  - Legibilidade
  - Criação de operadores sem sentido

Conversões de tipo

- Estreitamento
  - Converte um valor para um tipo que não pode armazenar aproximações equivalentes
  - double para float em Java
- Alargamento
  - int para float
- Decisão de projeto: expressões de modo misto
  - Convenções para conversão de tipo (coerção)
- Conversões explícitas: *casts*

Erros em Expressões

- Causas
  - Erros de Tipo
  - Limitações herdadas da aritmética
    - Ex: divisão por zero
  - Limitações da aritmética computacional
    - Ex: overflow, underflow, precisão
- Em geral são erros de execução que quando são detectados são exibidos na forma de exceções

Expressões relacionais

- Operador relacional: compara os valores dos seus dois operandos e retorna algum valor booleano
- Sobrecarregados
- Ex:
  - Diferente: != em C, /= em Ada, ~= em Lua, <> em JS
  - === previne PHP de fazer coerção
    - Ex: "7"==7 (V) mas "7"===7 (F)
- Precedência menor que os operadores aritméticos
  - `a+1 > 2*b`

Expressões booleanas

- AND, OR, NOT, XOR, NOR
- Em C o tipo int é usado para representar booleanos
  - 0 (zero) é falso
  - Diferente de 0 (zero) é true
- a > b > c em C é permitido?

Avaliação em curto-circuito

- Uma expressão no qual o resultado é determinado sem avaliar todos os operandos e/ou operadores
- Ex: `(13 * a) * (b / 13 - 1)`
  - Se `a` é zero, não há necessidade de avaliar `(b /13 - 1)`

```
if((a> b) ||((b++)))
     printf("Teste a=%d e b=%d",a,b);
```

Qual o resultado quando a=2 e b=3? E com os valores invertidos?

Comandos de Atribuição

- Uma das construções centrais do paradigma imperativo
- Atribuição simples
- Atribuição composta
  - `a = a + b => a += b`
- Atribuição unária
  - `a++`
- Dentro de expressões
  - `while((ch=getchar()) !=EOF){...}`
  - Efeito colateral

Atribuição Múltipla

- Perl, Ruby e Lua permitem múltiplos destinos e múltiplas origens em comandos
  - `($first, $second, $third) = (20, 30, 40);`
- Isto também é permitido na linguagem
  - `($first, $second) = ($second, $first);`
  - O que deve acontecer?

Atribuição em linguagens funcionais

- Identificadores em linguagens funcionais são só nomes de valores, não variáveis
- ML
  - Nomes são ligados a valores com **`val`**

```
val fruit = apples + oranges;
```

  - Se outro valor para fruit for definido, ele será um valor novo e diferente

Exercícios

- Defina o conceito de coerção
- Indique a ordem de avaliação das seguintes expressões:
  - a. a * b - 1 + c
  - b. a * (b - 1) / c mod d
  - c. (a - b) / c & (d * e / a - 3)
  - d. -a or c = d and e
  - e. a > b xor c or d <= 17
  - f. -a + b

Exercícios

- Dado o codigo a seguir

```
int fun(int *k) {
      *k += 4;
      return 3 * (*k) - 1;
}

void main() {
      int i = 10, j = 10, sum1, sum2;
      sum1 = (i / 2) + fun(&i);
      sum2 = fun(&j) + (j / 2);
}
```

- Quais são os valores de sum1 e sum2
  - Se os operandos são avaliados da esquerda para a direita?
  - Se os operandos são avaliados da direita para a esquerda?

Leitura Adicional

- Capítulo 7 – Expressões e Comandos de atribuição. SEBESTA, R. W. Conceitos de Linguagens de Programação. 9ª ED. BOOKMAN, 2011.
- Próxima aula
  - Capítulo 8 – Estruturas de Controle. SEBESTA, R. W. Conceitos de Linguagens de Programação. 9ª ED. BOOKMAN, 2011