



Linguagens de Programação

Aula 8

Celso Olivete Júnior

`olivete@fct.unesp.br`

Na aula passada

- ❑ Os tipos de dados de uma linguagem são uma grande parte do que determina o estilo e a utilidade de uma linguagem
- ❑ Os tipos de dados primitivos da maioria das linguagens imperativas incluem os tipos numéricos, de caracteres e booleanos
- ❑ Os tipos de enumeração e de subfaixa definidos pelo usuário são convenientes e melhoram a legibilidade e a confiabilidade dos programas
- ❑ Matrizes fazem parte da maioria das linguagens de programação
- ❑ Ponteiros são usados para lidar com a flexibilidade e para controlar o gerenciamento de armazenamento dinâmico

Na aula de hoje

Expressões e sentenças de
atribuição

Roteiro

1. Introdução
2. Expressões aritméticas
3. Operadores sobrecarregados
4. Conversões de tipos
5. Expressões relacionais e booleanas
6. Avaliação em curto-circuito
7. Sentenças de atribuição
8. 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

- ❑ 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

1. Quais são as regras de **precedência de operadores**?
2. Quais são as regras de **associatividade de operadores**?
3. Qual é a **ordem de avaliação** dos operandos?
4. Quais são os **efeitos colaterais na avaliação** de operandos?
5. A linguagem permite a **sobrecarga de operadores** definida pelo usuário?
6. 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 aritméticas

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

- ❑ Exemplo: $a + b * c \rightarrow$ supondo $a = 3$; $b=4$ e $c = 5$
 - Se a expressão for avaliada da direita para a esquerda: 23
 - Se avaliada da esquerda para a direita: 35

Expressões aritméticas

regras de precedência de operadores

□ Níveis de precedência mais utilizado
(hierarquia de prioridades)

1. parênteses
2. operadores unários
3. ** (exponenciação) → quando apresentar
4. *, /
5. +, -

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*
- ❑ Exemplo:
 - $a - b + c - d \rightarrow +$ e $-$ têm o mesmo nível de precedência, como determinar?

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**
- ❑ Exemplo:
 - $a - b + c - d \rightarrow + e -$ têm o mesmo nível de precedência, como determinar?
 - **É determinado por regras de associatividade**

Expressões aritméticas

regras de associatividade de operadores

- ❑ 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)

- ❑ Regras de precedência e associatividade podem ser alteradas com o uso de parênteses.

- ❑ Exemplo: $(a+b)*c$ → neste caso, a adição será avaliada primeiro

Expressões aritméticas

regras de associatividade de operadores

❑ Regras de associatividade

Linguagem	Regra de associatividade
Ruby	Esquerda: * / + - Direita: **
Ling. baseadas em C	Esquerda: * / % + - Direita: ++ e --

Expressões aritméticas

regras de associatividade de operadores

❑ Regras de associatividade

- Em **APL** (ling. destinada a operações matemáticas) a regra de associatividade é determinada da direita para esquerda
- Exemplo: **A * B + C**
 - O operador de adição é avaliado primeiro, seguido pelo operador de multiplicação

Expressões aritméticas

expressões em *Ruby*

- ❑ *Ruby* é uma linguagem orientada a objetos pura – todos os valores de dados são objetos
- ❑ Operadores aritméticos, relacionais e de atribuição, índices de matrizes, deslocamentos e operadores lógicos bit a bit, são implementados como métodos

Expressões aritméticas

expressões em *Ruby*

❑ Exemplo: **a + b**

- É uma chamado ao método **+** do objeto referenciado por **a**, passando o objeto referenciado por **b** como parâmetro

- ❑ Um resultado interessante da implementação de operadores como métodos é que esses operadores podem ser sobrescritos por programas de aplicação. Também podem ser redefinidos

Expressões aritméticas

expressões condicionais

❑ Expressões condicionais especificadas como sentença de atribuição

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

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

? Início do then

: Início do else

- Avalia como se fosse escrita como

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

Expressões aritméticas

ordem de avaliação de operandos

1. **Variáveis:** avaliadas a partir da obtenção dos seus valores a partir da memória
2. **Constantes:** algumas vezes avaliadas da mesma maneira. Em outros casos, é parte da instrução de linguagem de máquina e não requerer uma busca em memória

Expressões aritméticas

ordem de avaliação de operandos

3. **Expressões entre parênteses:** avaliam todos os operadores que ela contém antes de seu valor poder 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 (uma variável global é declarada fora da função, mas é acessível na função)

- ❑ Problema com *efeitos colaterais funcionais*:
 - Quando uma função referenciada em uma expressão altera outro operando da expressão; por exemplo, para uma mudança de parâmetro:

Expressões aritméticas

efeitos colaterais

❑ Exemplo:

```
a = 10;
```

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

❑ Supondo que **fun** retorna 10 e modifica o valor do seu parâmetro para 20

- Então se o valor de **a** for obtido primeiro (no processo de avaliação da expressão), seu valor é 10 e o valor da expressão **b** é 20
- Se o segundo operando (fun(a)) for avaliado primeiro, o valor do primeiro operando é 20 e o valor da expressão **b** é 30

Expressões aritméticas

efeitos colaterais

❑ Exemplo 2:

```
int a = 5;
int fun1() {
    a = 17;
    return 3
}
void main() {
    a = a + fun1();
}
```

❑ O valor calculado para **a** em **main** depende da ordem de avaliação dos operandos na expressão **a + fun1()**

- O valor de **a** será 8 se **a** for avaliado primeiro

Expressões aritméticas

efeitos colaterais

❑ Exemplo 2:

```
int a = 5;
int fun1() {
    a = 17;
    return 3
}
void main() {
    a = a + fun1();
}
```

- ❑ O valor calculado para **a** em **main** depende da ordem de avaliação dos operandos na expressão **a + fun1()**
 - O valor de **a** será 8 se **a** for avaliado primeiro
 - O valor de **a** será 20 se a chamada a função for avaliada primeiro

Expressões aritméticas

efeitos colaterais

❑ Duas possíveis soluções para o problema

1. Escrever a definição de linguagem para proibir efeitos colaterais funcionais
 - Não permitir parâmetros de duas direções
 - Não permitir 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 operações com `int` e `float`)
- ❑ Alguns são problema em potencial (por exemplo, & em C e C++)
 - Como um operador binário ele especifica uma operação lógica E (*AND*)
 - Como um operador unário seu significado é o endereço de uma variável.
 - Ex: **`x = &y;`**
 - ✓ 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`
 - ❑ Exercício: tanto o tipo `int` quanto o tipo `float` são armazenados utilizando 32 bits. Realize uma operação de alargamento (`int` para `float`) e verifique se ocorre a perda de dígitos de precisão

Conversões de tipos

expressão de 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

Conversões de tipos

modo misto

- ❑ Ex: *expressões de modo misto* na linguagem Java

```
int a;
```

```
float b, c, d
```

```
...
```

```
d = b * a;
```

- ❑ Essa expressão é permitida em Java, logo o compilador não consegue detectar o erro (c ao invés do a, por exemplo)
 - É inserido um código para realizar a coerção (implícita) do operando **a** em float.

Conversões de tipos

explícita

- ❑ Chamadas de *cast* em linguagens baseada em C

- ❑ Exemplo
 - C: `(int)angle`

 - ✓ Usa-se o parênteses no **tipo** devido às primeiras versões de C - apresentava nomes de **tipos** com duas palavras. Exemplo: **long int**

Conversões de tipos

Erros em expressões

❑ Causas

- Limitações inerentes da aritmética por exemplo, divisão por zero
- Limitações da aritmética computacional por exemplo, transbordamento (*overflow*) ou transbordamento negativo (*underflow*)

❑ Esses erros são detectados em tempo de execução do sistema - **exceções**

Expressões relacionais e booleanas

□ Expressões relacionais

- Um operador relacional compara os valores de seus dois operandos **(op1 oprel op2)**
- Uma expressão relacional tem dois operandos e um operador relacional
- O valor de uma expressão relacional é booleano
- Símbolos de operação variam um pouco entre as linguagens. Operadores de desigualdade: **!=, / =, ~ =, .NE., <>**

Expressões relacionais e booleanas

□ Expressões booleanas

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

FORTRAN 77	FORTRAN 90	C	Ada
<code>.AND.</code>	<code>and</code>	<code>&&</code>	<code>and</code>
<code>.OR.</code>	<code>or</code>	<code> </code>	<code>or</code>
<code>.NOT.</code>	<code>not</code>	<code>!</code>	<code>not</code>

Expressões relacionais e booleanas sem tipo booleano em C

- ❑ C 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`**)
 - **`b < c`** nunca é feito nessa expressão

Avaliação em curto-circuito

❑ Uma avaliação em curto-circuito é 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 $(b / 13 - 1)$

❑ Exemplo: $(a >= 0) \ \&\& \ (b < 10)$

Se a é menor que zero, não há necessidade de avaliar $(b < 10)$

Avaliação em curto-circuito

- ❑ Problema com avaliação que não é em curto-circuito. Ex: busca elemento em uma tabela

```
index = 1;
```

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

```
    index++;
```

- Se a avaliação não fosse em curto-circuito
 - 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 `||`) → E e OU

Sentenças de atribuição

- ❑ **Sentenças de avaliação** é uma das construções centrais em LP's imperativas → mecanismo onde o usuário pode mudar dinamicamente as vinculações de valores a variáveis
- ❑ A sintaxe geral
`<target_var> <assign_operator> <expression>`
- ❑ O operador de atribuição
 - `=` FORTRAN, BASIC e linguagens baseadas em C
 - `:=` ALGOL, 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 de uma unidade, adicionado a sum**)

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

`count++` (**count incrementado de uma unidade**)

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

- Obs: quando dois operadores são aplicados ao mesmo operando, a associação é feita da direita para esquerda

Sentenças de atribuição

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`

Sentenças de atribuição

Atribuição como uma expressão

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

- ❑ Um exemplo:

```
soma = contador = 10;
```

`contador=10` é realizado; o resultado é atribuído a `soma`

Sentenças de atribuição

Atribuição como uma expressão

- ❑ Em C, C++ e Java, a sentença de atribuição produz um resultado e pode ser usada como operandos
- ❑ Esse mecanismo ocasiona uma perda da detecção de erros.
- ❑ Um exemplo considerando a ling. C:
`if (x=y) ...` ao invés de `if (x==y) ...;`
 - Não gera erro: o valor de **y** é adicionado a **x** e então **x** é testado. O resultado gerado é diferente do esperado
 - Java e C# permitem apenas expressões booleanas em sentenças IF, logo esse problema não ocorre

Sentenças de atribuição

Atribuições de listas

- ❑ Perl e Ruby suportam atribuições de lista por exemplo,

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

Sentenças de atribuição

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

Exercícios

- Questões de revisão
 - 1, 8, 9, 10, 12 e 18
- Conjunto de problemas
 - 3, 4, 7, 9, 10 e 13
- Exercícios de programação
 - 1, 3, 8 e 9