

Expressões e Instruções de Atribuição

Prof. Dr. Eduardo Takeo Ueda
eduardo.tueda@sp.senac.br

Introdução

- Expressões são o meio fundamental de especificar computações em uma linguagem de programação:
 - Familiarização com as ordens de avaliação de operadores e de operandos:
 - Regras de associatividade e de precedência da linguagem.
 - A essência das linguagens de programação imperativas é o papel predominante das instruções de atribuição cuja finalidade é mudar o valor de uma variável.

Expressões Aritméticas (1/2)

- Sua avaliação foi uma das motivações para o desenvolvimento das primeiras linguagens de programação.
- Expressões aritméticas consistem de operadores, operandos, parênteses e chamadas de funções:
- Operadores podem ser:
 - **Unários** – um operando (-).
 - **Binários** – dois operandos (-, +, *, /).
 - **Ternário** – três operandos (? :).

Expressões Aritméticas (2/2)

- Questões de projeto para expressões aritméticas:
 - Quais são as regras de precedência para os operadores?
 - Quais são as regras de associatividade dos operadores?
 - Qual é a ordem de avaliação dos operandos?
 - Existem restrições nos efeitos colaterais da avaliação dos operandos?
 - A linguagem permite sobrecarga de operadores definidos pelo usuário?
 - Mistura de modos é permitida nas expressões?

Precedência de Operadores

- As regras de **precedência de operadores** para avaliação de expressões definem a ordem na qual operadores de diferentes níveis de precedência são avaliados.

FORTRAN	Pascal	Ada	C	Java
**	*, /, div, mod	**, abs	++, -- pós-fixado	++, -- pós-fixado
*, /	todos +, -	*, /, mod	++, -- pré-fixado	++, -- pré-fixado
todos +, -		+, - unário	+, - unário	+, - unário
		+, - binário	*, /, %	*, /, %
			+, - binário	+, - binário

$A + (-B) * C$ é válido!

$A + -B * C$ não é válido!

Associatividade (1/2)

- As regras de **associatividade** dos operadores para a avaliação da expressão definem a ordem na qual operadores adjacentes com o mesmo nível de precedência são avaliados:
- Regras de associatividade típicas:
 - Esquerda para a direita, exceto ****** que é da direita para a esquerda.
 - **APL** é diferente:
 - Todos os operadores tem precedência igual e todos os operadores associam da direita para a esquerda.

Associatividade (2/2)

FORTRAN

Esquerda: *, /, +, -
Direita: **

Pascal

Esquerda: todos

C

Esquerda: ++ pós-fixado, - pós-fixado, *, /, %, + binário, - binário
Direita: ++ pré-fixado, - pré-fixado, + unário, - unário

C++

Esquerda: *, /, %, + binário, - binário
Direita: ++, --, - unário, + unário

Ada

Esquerda: todos, exceto **
Não-associativo: **

Parênteses

- Os programadores podem alterar as regras de precedência e de associatividade colocando **parênteses** nas expressões:
- Linguagens que permitem parênteses em expressões aritméticas poderiam dispensar todas as regras de precedência e simplesmente associar todos os operadores da esquerda para a direita ou vice-versa:
 - Opção feita na linguagem de programação **APL**.

Expressões Condicionais

- Operador ternário, que faz parte do C, C++ e do Java:
 - condição ? expressão 1 : expressão 2

```
if (count == 0)
    media = 0;
else
    media = soma / count;
```

```
media = (count == 0) ? 0 : soma / count;
```

Ordem de Avaliação de Operandos

- Variáveis:
 - Simplesmente apanha o valor da variável.
- Constantes:
 - Algumas vezes apanha o valor da memória.
 - Algumas vezes a constante está numa instrução da linguagem de máquina.
- Expressões com parênteses:
 - Primeiro avalia todos os operandos e operadores.
- Funções por referência:
 - O caso de maior interesse, pois a ordem da avaliação é crucial.

Efeitos Colaterais (1/2)

- Um **efeito colateral** de uma função, chamado **efeito colateral funcional**, ocorre quando ela modifica um de seus parâmetros ou uma variável global:

```
int a = 5;
```

```
int fun() {  
    a = 17;  
    return 3;  
}
```

```
void main() {  
    int b = a + fun();  
}
```

Qual o valor de b?

8 ou 20?

Efeitos Colaterais (2/2)

- Duas possíveis soluções para o problema:
 - Escreva a definição da linguagem de modo a não permitir efeitos colaterais de funções:
 - Sem parâmetros de duas-vias em funções.
 - Sem referências não locais em funções.
 - Vantagem: funciona.
 - Desvantagem: programadores querem flexibilidade de parâmetros de duas vias e referências não locais.
 - Escrever a definição da linguagem requerendo que a ordem de avaliação dos operandos seja fixa:
 - Desvantagem: limita algumas otimizações de compilação.

Operadores Sobrecarregados (1/2)

- Utilizar o operador em mais de uma finalidade é chamado de **sobrecarga de operador** e geralmente é considerado aceitável, contanto que a legibilidade e/ou a confiabilidade não sofram:

```
int a = 10 + 5;  
String s = "Paradigmas" + "de" + "programação";  
  
double b = 7 / 2;  
double c = 7.0 / 2.0;
```

Operadores Sobrecarregados (2/2)

- Algumas são problemas em potencial:
 - `&` no `C`.
 - `*` no `C` e `C++`.
- Perda da detecção de erro de compilação:
 - Omissão de um operando deveria ser um erro detectável.
- Pode ser evitado pela introdução de novos símbolos:
 - `div` no `Pascal` para a divisão de inteiros.
- `C++` e `ADA` permite a sobrecarga de operadores definidos pelo usuário.
 - Problemas em potencial:
 - Usuários podem definir operações sem nexos.
 - Legibilidade pode ser prejudicada.

Conversões de Tipo

- Uma **conversão particularizante** é aquela que converte um objeto para um tipo que não pode incluir todos os valores do tipo original:
 - **double** para **float**, ou **double** para **int**.
- Uma **conversão generalizante** é aquela na qual um objeto é convertido para um tipo que pode incluir pelo menos aproximações para todos os valores do tipo original:
 - **float** para **double**, **int** para **double**.
 - São sempre seguras.

Conversões Implícitas

- Uma **coerção** é uma conversão de tipo implícita, ou seja, o compilador realiza automaticamente as conversões necessárias:
 - Desvantagem de coerções:
 - Diminuem a habilidade do compilador na detecção de erros de tipagem.
 - Na maioria das linguagens, todos os tipos numéricos são coagidos nas expressões, usando conversões generalizadas.
 - Em Modula-2 e ADA, praticamente ao existe coerção nas expressões.

```
int a, b;  
float c;  
a = b * c;
```


Conversões Explícitas

- Frequentemente chamadas **casts**:
 - ADA:
 - **FLOAT** (INDEX) – INDEX sendo do tipo INTERGER
 - C:
 - (int) speed /* speed sendo do tipo float */

Erros em Expressões

- Causados por:
 - Limitações inerentes da aritmética:
 - Divisão por zero.
 - Limitações da aritmética computacional:
 - *overflow*.
 - *underflow*.
 - Tais erros são muitas vezes ignorados pelo sistema em tempo de execução.

Expressões Relacionais (1/2)

- Um **operador relacional** compara os valores de seus dois operandos:
 - Uma **expressão relacional** tem dois operandos e um operador relacional.
 - Operadores relacionais sempre têm menor precedência do que os operadores aritméticos.

```
int c = a + 1 > 2 * b;
```

Expressões Relacionais (2/2)

Operação	Pascal	Ada	FORTRAN 77	C	Java
igual	=	=	.EQ.	==	==
diferente	<>	/=	.NE.	!=	!=
maior que	>	>	.GT.	>	>
menor que	<	<	.LT.	<	<
maior que ou igual	>=	>=	.GE.	>=	>=
menor que ou igual	<=	<=	.LE.	<=	<=

Expressões Booleanas (1/2)

- As expressões booleanas consistem em variáveis, em constantes, em expressões relacionais e em operadores booleanos, que normalmente incluem aquelas para operações AND, OR e NOT, e às vezes, para OR EXCLUSIVO e para equivalência:
 - C não possui o tipo booleano – usa o tipo `int` com 0 para falso e não zero para verdadeiro.
 - Uma característica estranha das expressões em C:
 - `a < b > c` é uma expressão legal, mas o resultado pode não ser o que se esperava.

Expressões Booleanas (2/2)

FORTRAN 77	FORTRAN 90	ADA	C	Java
.AND.	and	and	&&	&&
.OR.	or	or		
.NOT.	not	not	!	!

Avaliação Curto-circuito (1/2)

- Uma **avaliação curto-circuito** de uma expressão tem seu resultado determinado sem avaliar todos os operandos e/ou operadores:
 - **$(13 * A) * (B / 13)$**
 - Se A for ZERO, não existe a necessidade de avaliar a segunda expressão, pois $ZERO * X$ é sempre ZERO.
 - Normalmente os compiladores não detectando isso!
 - **$(A \geq 0) \text{ and } (B < 10)$**
 - Se a primeira expressão for FALSE, o compilador não precisa avaliar a segunda expressão, pois $FALSE \text{ and } X$ é sempre FALSE.

Avaliação Curto-circuito (2/2)

- **Pascal:**
 - Não usa avaliação curto-circuito.
- **C, C++ e Java:**
 - Usam avaliação curto-circuito para os operadores booleanos usuais (**&&** e **||**), mas também possuem operadores booleanos que não são curto-circuito (**&** e **|**).
- **ADA:**
 - Programador pode especificar qualquer dos dois (curto-circuito é especificado com **and then** e **or else**);.
- Avaliação curto-circuito expõe o problema potencial de efeitos colaterais nas expressões:
 - **(a > b) || (b++ / 3)**

Instruções de Atribuição (1/2)

- A sintaxe geral da instrução de atribuição simples é:
 - <variável_alvo> <operador_de_atribuição> <expressão>
 - Operadores de atribuição:
 - (=) → FORTRAN, BASIC, PL/I, C, C++, Java:
 - Pode ser ruim se sobrecarregado para o operador relacional de igualdade:
 - (PL/I) A = B = C;
 - (:=) → ALGOL, Pascal, Modula-2, ADA.

Instruções de Atribuição (2/2)

- Alvos múltiplos:
 - PL/I: `A, B = 10;`
 - C e C++: `A = B = C = 10;`
- Alvos condicionais (C, C++ e Java):
 - `(flag) ? count1 : count2 = 0;`
- Operadores de atribuição compostos (C, C++ e Java):
 - `sum += next;`
- Operadores de atribuição unários (C, C++ e Java):
 - `a++; -a++; sum = ++count;`
- C, C++ e Java tratam `=` como um operador aritmético binário:
 - `a = b * (c = d * 2 + 1) + 1;`

Atribuição como uma Expressão

- Em **C**, **C++** e **Java** o comando de atribuição produz um resultado:
 - Logo, eles podem ser usados como operandos em expressões:
 - `while ((ch = getchar()) != EOF) { }`
 - Desvantagem:
 - Outra espécie de efeito colateral da expressão.

Atribuição com Mistura de Modos

- Em **FORTRAN**, **C** e **C++** qualquer valor numérico pode ser atribuído a qualquer variável escalar numérica:
 - Qualquer conversão necessária é feita.
- Em **Pascal**, inteiros podem ser atribuídos a reais, mas reais não podem ser atribuídos a inteiros (o programador deve especificar se a conversão de real para inteiro é truncada ou arredondada).
- Em **Java** apenas coerção de atribuições generalizadas são permitidas.
- Em **ADA** não há coerção na atribuição.