Capítulo 7

Expressões e Sentenças de Atribuição

Tópicos

- Introdução
- Expressões aritméticas
- Sobrecarga de operadores
- Conversão de tipos
- Expressões booleanas e relacionais
- Avaliação em curto-circuito
- Declarações de alocação
- Alocação de modo misto

Introdução

- **Expressões**: meio fundamental para **especificar uma tarefa computacional** em uma linguagem de programação
- Para "**avaliar uma expressão**" é necessário estar familiarizado com a ordem das operações e com a avaliação de operandos
- Uma característica essencial das linguagens imperativas é o papel dominante de declarações de alocação

Expressões Aritméticas

- Avaliar expressões aritméticas foi um dos principais motivos para o desenvolvimento das primeiras linguagens de programação
- Expressões aritméticas consistem de:
  - Operandos
  - Operadores
  - Parênteses
  - Chamadas de funções

Expressões Aritméticas: projeto

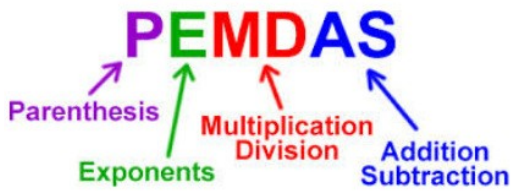
- Algumas questões de projeto importantes ao lidarmos com expressões aritméticas
  - Regras de **precedência**?
  - Regras de **associatividade**?
  - **Ordem de avaliação** dos operandos?
  - **Efeitos colaterais** na avaliação dos operandos?
  - **Sobrecarga** de operadores?
  - **Mistura de tipos** em expressões?

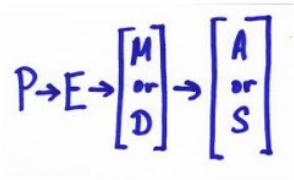
Expressões Aritméticas: operandos

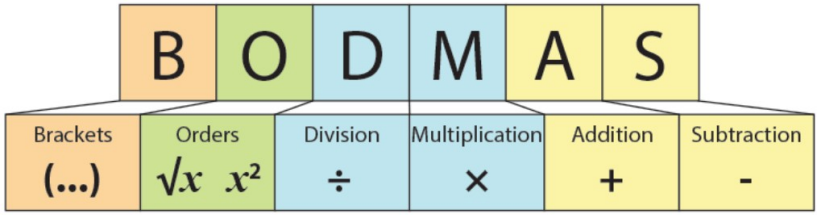
- Expressões **unárias**
- Expressões **binárias**
- Expressões **ternárias**

Expressões Aritméticas: precedência

- **Regras de precedência** definem a ordem na qual operadores de diferentes níveis são avaliados
- Basicamente segue a matemática:
  - Agrupamentos
  - Operadores unários
  - Potenciação, radiciação
  - Multiplicação e divisão
  - Adição e subtração







Expressões Aritméticas: associatividade

- Regras de associatividade definem a ordem na qual operadores de mesmo nível são avaliados
- Basicamente, segue a matemática:
  - **Esquerda**: *, /, +, -
  - **Direita**: ^

- Atenção: precedência e associatividade podem ser alteradas com parênteses

Expressões Aritméticas: condicionais

- Algumas linguagens suportam expressões aritméticas condicionais (C, C++, etc.)

```
media = (contagem == 0) ? 0 : soma / contagem
```

  - É como se estivesse escrito como:

```
if (contagem == 0):
  media = 0
else:
  media = soma / contagem
```

Expressões Aritméticas: ordem de avaliação dos operandos

- **Ordem de avaliação dos operandos**
  1. Variáveis: busca o valor da memória
  2. Constantes: busca o valor da memória (ou está nas instruções de máquina)
  3. Expressões com parênteses: avalia todos os operandos e operadores primeiro
  4. Se os operandos são chamadas de função, a coisa é um pouco mais complicada
     - Se nenhum dos operandos tem **efeito colateral**, a ordem de avaliação é irrelevante
     - Se algum dos operandos tem **efeito colateral**, deve-se tomar cuidado!

Expressões Aritméticas: efeitos colaterais

- **Efeito colateral**: ocorre quando uma função altera um de seus parâmetros ou uma variável não local. Pode ser fonte de muitos erros nos programas!

```
a + fun(a)

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

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

Efeitos Colaterais

- Existem duas possíveis soluções:
  1. A linguagem não permitir efeitos colaterais
     - **Desvantagem:** baixa flexibilidade e perda de referências não-locais
  2. A linguagem deve exigir que a ordem de avaliação dos operandos seja fixa
     - **Desvantagem**: limita otimizações pelo compilador
     - Ex.: Java obriga que seus operandos sejam avaliados em ordem da esquerda para direita

Transparência Referencial

- Se duas expressões em uma linguagem têm o mesmo valor e puderem ser substituídas entre si em qualquer local do programa, sem afetar a ação do programa

```
resultado1 = (fun(a) + b) / (fun(a) – c);
temp = fun(a);
resultado2 = (temp + b) / (temp – c);
```

Se `fun` não tem efeitos colaterias, então

```
resultado1 = resultado2
```

Caso contrário, não tem transparência referencial

Transparência Referencial

- Vantagem:
  - Semântica do programa é mais fácil de entender
- Como linguagens funcionais puras não têm variáveis, possuvem transparência referencial
  - Funções não tem estado, que seriam armazenados em variáveis locais
  - Se uma função utilzia um valor não-local, deve ser uma constante, então o valor de uma função depende apenas de seus parâmetros

Sobrecarga de Operadores

- Uso de um operador para mais de um propósito
- Comum:
  - Operador + para **`int`** e **`float`**
- “Problemáticos”:
  - Operador * no C e C++

Sobrecarga de Operadores

- Algumas linguagens permitem que o usuário definam seus próprios operadores com sobrecarga
  - Se usado com sabedoria: bom! Evitam chamadas de métodos e as expressões parecem naturais.
  - Se usado com descuido:
    - Usuários podem definir operações sem sentido

Conversão de Tipos

- Conversão de estreitamento:
  `float` para **`int`**
- Conversão de alargamento:
  `int` para **`float`**

Conversão de Tipos: modo misto

- Expressão de modo misto: é uma expressão que tem **operandos de diferentes tipos**. Para funcionar a linguagem deve suportar conversão de tipos, que pode ser:
  - Coerção (implícito)
  - Cast (explícito)

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

```
(int) angle
```

Erros nas Expressões

- Causas
  - Aritméticas:
    - Divisão por zero
  - Limitações do computador:
    - Overflow
    - Underflow

Expressões relacionais e booleanas

- Expressões Relacionais:
  - Usam operadores relacionais e operandos de diversos tipos
  - Avaliam para alguma representação booleana
  - Operadores variados (`!=`, `/=`, `~=`, `.NE.`, `<>`)

- Expressões Booleanas:
  - Usam operandos booleanos
  - Avaliam para booleanos

Expressões relacionais e booleanas

Highest	postfix ++, --
	unary +, unary -, prefix ++, --, !
	*, /, %
	binary +, binary -
	<, >, <=, >=
	=, !=
	&&
Lowest	||

Avaliação em curto-circuito

- O resultado de uma expressão é determinado ANTES de avaliar todos os operandos e/ou operadores

```
(13 * a) * (b / 13 - 1)
```

```
(a >= 0) && (b < 10)
```

A and B and C and D and E

A or B or C or D or E

Sentenças de Atribuição: simples

- Sintaxe geral:

  `<var_alvo> <operator_atribução> <expressão>`

- O operador de atribuição muda de acordo com a linguagem
  - = C e derivados
  - := PL/SQL

Sentenças de Atribuição: alvos condicionais

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

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

Sentenças de Atribuição: operadores compostos

```
sum += value;
sum = sum + value;
```

Sentenças de Atribuição: operadores unários

```
sum = ++ count;
```

```
sum = count ++;
```

Sentenças de Atribuição: operadores unários

```
sum = ++ count;
```

```
count = count + 1;
sum = count;
```

```
sum = count ++;
```

```
sum = count;
count = count + 1;
```

Sentenças de Atribuição: atribuição como expressão

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

Sentenças de Atribuição: atribuição múltipla

```
($first, $second, $third) = (20, 40, 60);
```

```
($first, $second) = ($second, $first);
```

Até a próxima!