FACAMP
FACULDADES DE CAMPINAS

Linguagens de Programação

Aula 7

Expressões, atribuição e estruturas de controle

2º semestre de 2019

Prof José Martins Junior

Expressões

- São os meios fundamentais de especificar computações em uma linguagem de programação
- Conceitos envolvidos
  - Ordens de avaliação de operadores e operandos, ditadas pelas regras de associatividade e de precedência da linguagem
  - Diferenças e conversões de tipos e avaliação em curto-circuito
- Questões de projeto
  - Quais são as regras de precedência de operadores?
  - Quais são as regras de associatividade de operadores?
  - Qual é a ordem de avaliação dos operandos?
  - Existem restrições acerca de efeitos colaterais na avaliação de operandos?
  - A linguagem permite a sobrecarga de operadores definida pelo usuário?
  - Que tipo de mistura de tipos é permitida nas expressões?

Expressões aritméticas

- Em linguagens de programação consistem em operadores, operandos, parênteses e chamadas a funções
- Um operador pode ser
  - Unário por possuir um único operando
  - Binário, dois operandos
  - Ternário, três operandos
- Notação infix (convencional)
  - Operadores binários entre seus operandos
  - Usada na maioria das linguagens imperativas
- Notação prefix (polonesa)
  - Operadores prefixados, precedendo seus operandos
  - Exemplo de linguagem: Perl

Ordem de avaliação de operadores

- Precedência
  - O valor de uma expressão depende ao menos parcialmente da ordem de avaliação dos operadores na expressão
  - Geralmente, baseia-se em sentido (esquerda para direita, p.e) e na hierarquia (quem é avaliado antes) dos operadores
  - Operadores unários
    - **+** operador identidade (Java e C#: conversão implícita short/byte para int)
    - **-** modifica o sinal do operador; no meio de expressões, requer parênteses
  - Hierarquia de operadores em algumas linguagens

    ** é exponenciação em Ruby; rem em Ada é o mesmo que % em C

	Ruby	*Linguagens Baseadas em C*	*Ada*
Mais alta	**	++ e - - pós-fixados	**, **abs**
	+ e – unários	++ e - - pré-fixados, + e – unários	*, /, **mod**, **rem**
	*, /, %	*, /, %	+ e – unários
Mais baixa	+ e – binários	+ e – binários	+ e – binários

Ordem de avaliação de operadores

- Associatividade
  - Regras da linguagem que definem a ordem na avaliação de operadores de mesmo nível de precedência
  - Associatividade à esquerda
    - Forma mais utilizada nas LP imperativas. Ex.: a – b + c
    - Ocorrência mais à esquerda é avaliada primeiro
  - Associatividade à direita
    - Ocorrência mais à direita é avaliada primeiro
    - Mais incomum, mas ocorre com operadores unários em Java e C, e também com a exponenciação em Ruby e Fortran

Linguagem	*Regra de Associatividade*
Ruby	Esquerda: *, /, +, - Direita: **
Linguagens Baseadas em C	Esquerda: *, /, %, + binário, - binário Direita: ++, --, - unário, + unário
Ada	Esquerda: Todos, exceto ** Não associativo: **

Ordem de avaliação de operadores

- Parênteses
  - Permitem alterar as regras de precedência e de associatividade em expressões
  - Uma parte com parênteses de uma expressão tem precedência em relação às suas partes adjacentes sem parênteses
  - Os parênteses podem ser aninhados e a ordem de avaliação será sempre de “dentro para fora”
  - As linguagens de programação geralmente não utilizam de outros símbolos (chaves e colchetes) para tal fim, como na Matemática

Expressões condicionais

- Forma comprimida de expressar uma expressão if-then-else
  - Presente em muitas linguagens (incluindo C, Java)
  - Forma: **expressão_1 ? expressão_2 : expressão_3**
    - Primeira parte: teste booleano
    - Sinal ? em expressões condicionais é um operador **ternário**
  - Pode ser útil em sentenças de atribuição, como

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

    - count será comparado com zero
    - Se for verdade, atribui 0
    - Senão, atribui o resultado da expressão sum / count

Efeitos colaterais funcionais

- Efeito colateral de uma função ocorre quando ela modifica um de seus parâmetros ou uma variável global
  - Exemplo em C:

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

  - O valor de **a** em main depende da ordem de avaliação dos operandos
    - Será 8, se **a** for avaliado primeiro
    - Ou 20, se a chamada a função for avaliada primeiro

Transparência referencial

- Propriedade de um programa se quaisquer duas expressões, com o mesmo valor, puderem ser substituídas uma pela outra em qualquer lugar no código, sem afetar a sua ação
  - Pode ser afetada pelos efeitos colaterais funcionais
  - Exemplo:

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

    - Se a função fun não tem efeitos colaterais, result1 e result2 serão iguais
    - Entretanto, suponha que fun tem o efeito colateral de adicionar 1 a b ou a c. Então result1 não seria igual a result2
  - Como não têm variáveis, os programas escritos em **linguagens funcionais** puras são transparentes referencialmente

Sobrecarga de operadores

- Uso de um mesmo operador para múltiplos propósitos
  - Exemplos:
    - O operador + realiza soma de valores numéricos, porém, em Java, é também utilizado para concatenar Strings
    - O operador & realiza operação lógica E bit a bit mas, em C/C++, quando colocado antes de uma variável, referencia seu endereço

Expressões relacionais

- Combinam operandos com operadores relacionais
  - Operador relacional compara os valores de seus dois operandos
  - O resultado de uma expressão relacional é booleano
  - Operadores relacionais de igualdade e desigualdade em C e Java

    == (igual) != (diferente) < (menor) > (maior)

    <= (menor ou igual) >= (maior ou igual)
  - Diferença entre operadores **!=** (C e Java) em outras linguagens

    Ada usa **/=**, Lua usa **~=**, Fortran 95 usa **.NE.** ou **<>**
  - JavaScript e PHP têm operadores === e !== que não impõem coerção
    - Exemplos em JavaScript:

      "7" == 7 (verdade, pois JS converte o primeiro em número, por coerção)

      "7" === 7 (falso, pois o operador previne a coerção)

Expressões booleanas

- Consistem na combinação de operandos booleanos e expressões relacionais com operadores booleanos
  - Operadores geralmente incluem E, OU, NOT, XOR, como em C/Java

    & (and) | (or) ^ (xor) ~ (not)
  - Operadores curto-circuito, como em C/Java

    && (and) Se 1° operando for false o 2° não é calculado

    || (or) Se 1° operando for true o 2° não é calculado
  - Precedências em C

Mais alta	++ e --pós-fixados
	+ e – unários, ++ e -- pré-fixados, !
	*, /, %
	+ e – binários
	<, >, <=, >=
	=, !=
	&&
Mais baixa	||

Sentenças de atribuição

- Representam a atribuição de valor a uma variável (à esquerda)
  - Maioria das linguagens usa o sinal de igualdade

    = em C, Java e outras

    := em Pascal, Algol
  - Atribuição composta ou atalho

    += -= *= /= %=
    - Exemplo: `x += 1;`
    - Equivalente a: `x = x + 1;`
  - Operador de atribuição unário

    ++ (incremento) -- (decremento)
    - Exemplo: `x = 1;`

      `y = x++;` (y receberá 1, e x receberá 2)

Atribuição como uma expressão

- Linguagens baseadas em C, a sentença de atribuição produz um resultado, que é o mesmo que o valor atribuído ao alvo
  - Pode-se utilizar o resultado atribuído como operando em uma expressão

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

  - A atribuição pode ser feita em cadeia

```
sum = count = 0;
```

    - Primeiro count recebe o valor zero, depois o valor de count é atribuído a sum
  - Problema com a detecção de erros em C
    - Se, por engano, um programador digitar

```
if (x = y) ...
```

      em vez de

```
if (x == y) ...
```

    - Caso o valor de y seja diferente de 0 (falso em C), mas diferente de x, a primeira expressão resultaria em verdade

Atribuição em lista

- Algumas linguagens, como Perl, Ruby, Lua e Matlab, permitem atribuir múltiplos valores ao mesmo tempo
  - Exemplo em Perl

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

  - E uma troca poderia ser feita na forma

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

Precedência de operadores em Java

[] () chamada de método	esquerda para direita
! ~ ++ -- +(unário) -(unário) Casting new	direita para esquerda
* / %	esquerda para direita
+ -	esquerda para direita
<< >> >>>	esquerda para direita
< <= > >= instanceof	esquerda para direita
== !=	esquerda para direita
&	esquerda para direita
^	esquerda para direita
|	esquerda para direita
&&	esquerda para direita
||	esquerda para direita
?:	esquerda para direita
= += -= *= /= %= &= |= ^= <<= >>= >>>=	direita para esquerda

Estruturas de controle

- Uma estrutura de controle é uma sentença de controle e a coleção de sentenças cuja execução ela controla
- As principais estruturas de controle são
  - Sentenças de seleção
  - Sentenças de iteração
  - Desvio incondicional

Sentenças de seleção

- Uma sentença de seleção oferece os meios de escolher entre dois ou mais caminhos de execução em um programa
- Duas categorias gerais
  - Seleção bidirecional
  - Seleção n-dimensional ou múltipla

Sentenças de seleção bidirecional

- Permitem escolher entre dois ou mais caminhos de execução em programa
  - Forma geral

    if (expressão booleana)

    instrução

    Else instrução

Formato em C/Java:

```
if (condição) {
        instruções; ...
}
    [ else {
        instruções; ...
} ]
```

  - Linguagens como Fortran possuíam forma unidirecional (com GOTO)

```
IF (.NOT. condição) GOTO 20
        I  = 1
        J = 2
20 CONTINUE
```

Sentenças de seleção aninhadas

- Em Java, o else é associado ao if mais próximo

```
if (sum == 0)
  if (count == 0)
       result = 0;
       else result = 1;
```

- Em Pascal, o else é associado ao then mais próximo

```
if ... then
  if ... then
...
  else ...
```

- Para forçar, pode-se usar uma instrução composta, como em C

```
if (sum == 0) { if (sum == 0) {
 if (count == 0) result = 0;
}
else result = 1;
```

Sentenças de seleção múltipla

- Permite a seleção de uma instrução, dentre qualquer número de instruções ou de grupos de instruções
- Duas formas geralmente presentes em LPs
  - Seletores múltiplos modernos
  - Seletores múltiplos usando if

Seletores múltiplos modernos

- Pascal "case"
  - Sintaxe:

```
case expressão of
  constante_1 : instrução_1;
  ...
  constante_n : instrução_n
end
```

- C/Java “switch/case”
  - Sintaxe (em C, somente números inteiros; em Java, também Strings)

```
switch (escolha) {
        case valor: instruções; ... break;
        case valor: instruções; ... break;
        default: instruções;
}
```

Select case em VBA

- A linguagem permite uso de intervalos, não só igualdades
  - Exemplo

```
Dim Number
Number = 8    ' Initialize variable.
Select Case Number    ' Evaluate Number.
Case 1 To 5    ' Number between 1 and 5, inclusive.
   Debug.Print "Between 1 and 5"
' The following is the only Case clause that evaluates to True.
Case 6, 7, 8    ' Number between 6 and 8.
   Debug.Print "Between 6 and 8"
Case 9 To 10    ' Number is 9 or 10.
Debug.Print "Greater than 8"
Case Else    ' Other values.
   Debug.Print "Not between 1 and 10"
End Select
```

Seletores múltiplos usando if

- Extensões diretas de seletores bidirecionais, usando cláusulas else-if
  - Por exemplo em Ada:

```
if ...
 then ...
elsif ...
 then ...
 else ...
end if
```

Exemplo em C/Java:

```
if (condição1) {
        instruções; ...
} else if (condição2) {
        instruções; ...
} else {
        instruções; ...
}
```

Sentenças de iteração

- Permitem repetir uma instrução ou um bloco de instruções
- Tipos de sentenças iterativas
  - Laços controlados por contador
  - Laços controlados logicamente
  - Laços controlados por iterador

Laços controlados por contador

- Possui uma variável do laço, e meios de especificar os valores inicial e terminal, e o tamanho do passo
- Instrução for do Pascal

```
for variavel := inicial (to | downto) final do
    comando
```

Formato em C/Java:

```
for (inicialização ; condição ; passo) {
        instruções; ...
}
```

- Instrução for em VBA

```
For contador = início To fim [ Step etapa ]
[ instruções ]
[ Exit For ]
[ instruções ]
Next [ contador ]
```

Laços controlados logicamente

- O controle da repetição baseia-se em uma expressão booleana e não em um contador
- Formatos em Pascal

Pré-teste (executa enquanto

```
while (condição) do
begin
        instruções ...
end
```

Pós-teste (executa a primeira vez e repete enquanto falso)

```
repeat
        instruções ...
until (condição)
```

- Formatos em C/Java

Pré-teste (executa enquanto verdadeiro)

```
while (condição) {
        instruções; ...
}
```

Pós-teste (executa a primeira vez e repete enquanto verdadeiro)

```
do {
        instruções; ...
} while (condição);
```

Laços controlados por iterador

- A quantidade de repetições do laço é definida pelo número de elementos de um conjunto ou de uma iteração
- Exemplo de instrução foreach em Perl

```
$nomes = {"José","João","Joca"};
foreach $nome (@nomes)
{
 print $nome
}
```

- Formato do for enhanced em Java

```
for(Elemento e: conjunto) {
    instruções; ...
}
```

Controle do laço pelo usuário

- Programador pode interromper a execução do laço
  - Em C , C++ e Java: break
    - Se chamado em um bloco de laço (contador, lógico ou iterador), interrompe e sai
  - Uma alternativa (em C, C++ e Java): continue
    - Pula o resto das instruções da iteração atual, mas não sai do laço

Desvio incondicional

- Uma instrução de desvio incondicional transfere o controle para outra posição do programa
- Instrução mais conhecida é **goto**
  - Seu uso pode causar problemas de legibilidade
  - Java não tem goto, mas possui uma forma de break rotulado
- Exemplo de "goto" em C

```
printf("Enter m for mesg, or e to end:");
scanf("%c",&letter);
if(letter=='m') goto A;
else  goto B;
A: printf("\nHello!, you pressed m");
goto FIM;
B: printf("\nBye!, ending program");
FIM:
```

Bibliografia

- SEBESTA, R. W. Conceitos de Linguagens de Programação. Porto Alegre: Bookman, 2011. 792p.