Expressões e Atribuição

Prof. Hugo de Paula

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Departamento de Ciência da Computação

Sumário

Expressões

- Expressões são a forma mais fundamental de computação em uma linguagem de programação.
- Deve-se levar em consideração a forma de avaliação, a ordem de precedência dos operadores e a avaliação dos operandos.

Expressões aritméticas

Expressões aritméticas consistem de operadores, operandos, parêntesis, e chamadas de funções.
Aspectos de projeto de expressões aritméticas incluem:

- Regras de precedência de operadores.
- Associatividade de operadores.
- Ordem de avaliação dos operandos.
- Efeitos colaterais.
- Sobrecarga de operadores.
- Combinação de tipos em expressões.

Precedência e Associatividade de operadores

Precedência: regras que definem a ordem em que operadores adjacentes são avaliados. Níveis típicos:

- parêntesis.
- operadores unários.
- ** (potência, se suportado).
- *, / e %.
- + e -.
- combinação de tipos em expressões.

Associatividade: regras que definem a ordem em que operadores adjacentes com a mesma precedência são avaliados. Regra típica:

- Da esquerda para a direita (exceto **).

Ordem de avaliação de operandos

1. Variáveis: inspeciona o valor da memória.
2. Constantes: inspeciona na memória, ou em alguns casos é uma instrução da linguagem de máquina.
3. Expressões em parêntesis.
4. Chamada de função.

Expressões com efeito colateral

- Além de produzir um valor, sua avaliação também altera o ambiente (atualiza variáveis não locais, ou parâmetros bidirecionais, como em passagem por referência).
- Exemplo: getchar(arq)
  - Retorna o caractere corrente de um arquivo
  - Efeito colateral: avançar a posição corrente de leitura/escrita no arquivo
  - Com isso, provavelmente o trecho de programa abaixo está incorreto

```
if (getchar(arq) == 'F')
     sexo= feminino;
else if (getchar(arq) == 'M')
     sexo= masculino;
```

Transparência referencial

Transparência referencial

Um programa tem a propriedade de transparência referencial, se quais duas expressões no programa que possuem o mesmo valor podem ser substituídas uma pela outra em qualquer lugar do programa, sem afetar o comportamento do mesmo.

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

Se fun não produz efeito colateral, então result1 = result2.
Caso contrário, a transparência referencial é violada.

Transparência referencial

- Semântica de um programa é mais fácil de entender se o programa possui transparência referencial.
- Por não possuírem variáveis, programas de linguagens puramente funcionais possuem transparência referencial.
  - funções não possuem estado.
  - Se uma função utiliza um valor externo a ela, este valor deve ser constante.
  - Desta forma, o valor da função depende apenas de seus parâmetros.

Conversão de tipos

- Conversões de estreitamento (*narrowing*), converte objeto para um tipo que não inclui todos os valores do tipo original. Exemplo: **float** para **int**.
- Conversões de alargamento (*widening*), converte objeto para um tipo que inclui pelo menos aproximações de todos os valores do tipo original. Exemplo: **int** para **float**.
- **Coerção**: conversão implícita de tipos.
  - Na maioria da LPs, tipos numéricos sofrem coerção através de conversões de alargamento.
- ***Type casting***: conversão explícita de tipos. Exemplos:
  - Em C: **float** numReal; **int** numInt = (**int**) numReal;

Regras de coerção da linguagem C#

De	Para
sbyte	short, int, long, float, double, decimal
Byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
Long	float, double, decimal
char	ushort, int, uint, long, ulong, float, double, decimal
float	double
ulong	float, double, decimal

Avaliação em curto circuito

- Uma expressão em que o resultado pode ser determinado sem que sejam avaliados todos os operandos e/ou operadores.
- Exemplos:
  - (13 * a) * (b / 13 - 1)

    Se a é 0, então não é necessário avaliar (b /13 - 1).
  - (ano % 4 == 0) && ((ano % 100 != 0) || (ano % 400 == 0))

    Se ano não é múltiplo de 4 (75% das vezes), o ano não é bissexto, independente das outras condições.
- Falta da avaliação em curto circuito pode causar erros:

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

  Pode causar erro de índice fora da faixa se valor inicial de index for maior que o tamanho da lista.

Avaliação em curto circuito

- C, C++ e Java usam avaliação em curto circuito para operadores booleanos, mas não oferecem curto circuito para operadores lógicos bit a bit.
- Curto-circuito podem causar problemas em conjunto com efeito colateral:

```
(a > b)  ||  (b++ / 3)
```

Comandos

- **Comando**: estrutura sintática cuja execução atualiza variáveis.
- Característicos de linguagens imperativas.
- Principais comandos de uma linguagem imperativa:
  - Atribuição
  - Chamada de Procedimento
  - Sequenciais
  - Condicionais
  - Iterativos
  - Entrada e saída

Atribuição

VARIAVEL ← EXPRESSAO

acesso a variável <operador de atribuição> valor (expressão)

- se referências são valores de 1a. classe, variável é uma expressão do tipo referência.

Exemplo em ML

```
( if ... then m else n ) := 7
```

Exemplo em Perl

($flag ? $total : $subtotal) = 0 é equivalente a

```
if ($flag) $total = 0
else $subtotal = 0
```

Atribuição

- Atribuição múltipla.
  - m := n := 0;
  - m := n := <expr> é equivalente à m := <expr>; n := <expr>;?
- Atribuição simultânea.
  - m, n := n, m

Exemplo em Perl

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

- Combinação de operadores com atribuição.
  - m −= 10 m ∗= 30

Acesso a Variáveis

- Observe os códigos abaixo:
  - **read** (n); n := n + 1; **write** (n);
- **Quais os acessos a n ?**
- **O que significam?**
- Dois contextos possíveis
  - uma referência para a variável
  - o conteúdo da variável (valor)
- uma variável sempre produz uma referência (identificado é associado a uma declaração)
  - operação implícita substitui a referência pelo conteúdo (“dereferenciação”)
  - **read**(n); n := valor(n) + 1; **write** (valor(n));