Centro Tecnológico
Departamento de Informática

Prof. Veruska Zamborlini
veruska.zamborlini@inf.ufes.br
http://www.inf.ufes.br/~veruska.zamborlini

Aula 7
Expressões e Sentenças de Atribuição
2021/2

Material adaptado
Prof.s Vitor Souza e Eduardo Zambon

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Introdução

- **Expressões**: meios fundamentais de especificar computações em uma LP
  - *Exemplo*: 3 + 2.7

- **Sentenças de atribuição**: fundamental em LP imperativas!
- Modifica o valor de uma variável dada uma ***expressão***.
  - *Exemplo*: x = 3 + 2.7 ou x := 3 + 2.7

Expressões

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Expressões

- Meios de especificar atribuições/computações em uma LP
- Podem conter operadores e operandos
- Expressões simples: até um operador

  ‘a’ 3 + 2.7
- Expressões compostas: mais de um operador

  2*3^3 + 2.7

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

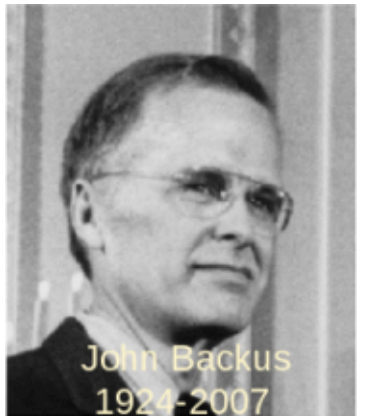
Expressões

- Crucial entender **sintaxe** e **semântica**
- **Sintaxe** (Capítulo 3 - será estudado em outras disciplinas)
  - Como pode-se formar uma expressão?
  - Exemplo: (*3 + 2*) ou +(3 2)
- **Semântica** (Capítulo 7)
  - O que uma expressão significa?
  - Como ela é avaliada? Ou, qual a ordem de avaliação dos operadores e operandos?

Sintaxe (prévia)

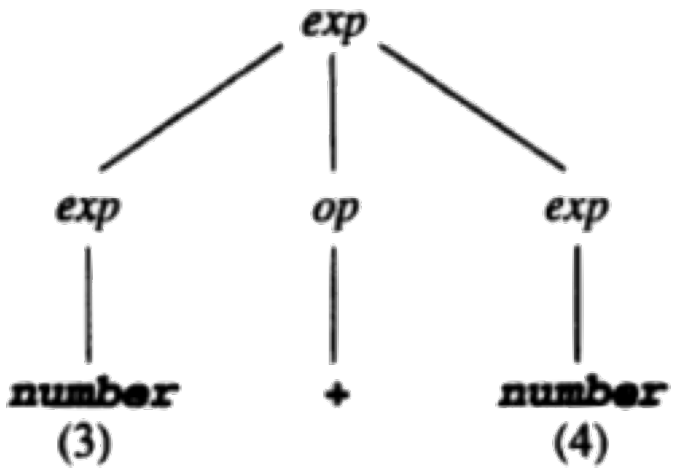
- ***Como pode-se formar uma expressão?***
- BNF (Backus-Naur Form) é a forma padrão de descrever sintaxe de LPs
- Foi usada por John Backus e Peter Naur para descrever a sintaxe de Algol 60.
- Notação original usa o meta-símbolo ::= e cerca os não-terminais por < e >

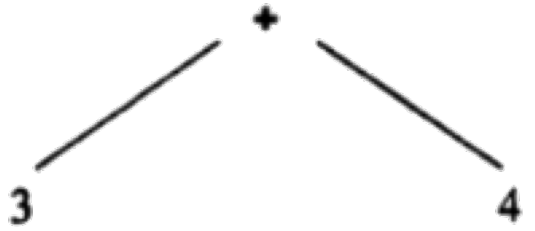
```
<exp> ::= <exp> <op> <exp> | (<exp>) | NUM
<op>  ::= + | - | * | /
```



Sintaxe (prévia)

- BNF dá origem a árvores de sintaxe (concreta e abstrata)
- AST (abstract syntax tree) ⇒ representação do programa





- BNF (gramática livre de contexto) e Árvores de Sintaxe serão estudadas em Linguagens Formais e Compiladores.

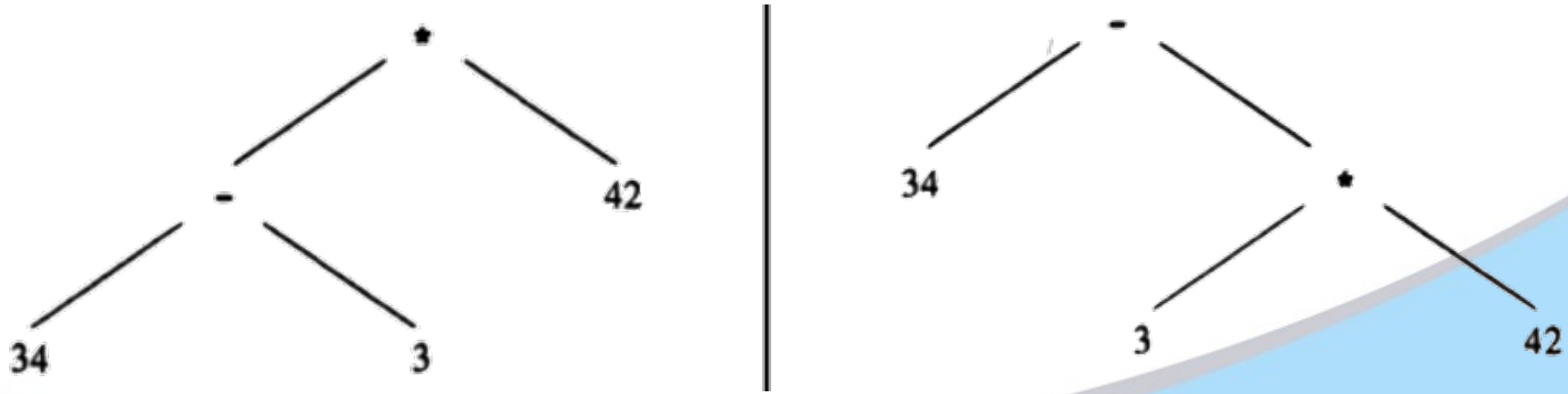
UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Operadores - Sintaxe

- Unários
  - - 1;
- Binários
  - 4 + 2;
- Ternários
  - y > 0 ? 4 : 2;
- Eneários:
  - varargs C/C++/ Java
- Pré-fixados (prefix) - LISP
  - (+ a (* b c))
- Infixados (infix)
  - a + (b * c)
- Pós-fixados (postfix)
  - b c * a +
  - Muito útil! Não precisa de parênteses, simula uma pilha

Semântica

- Diferença de sintaxe entre LPs é um tanto óbvia, mas a semântica não.
- Duas ASTs para a mesma entrada: 34 - 3 * 42
- Qual das duas é a correta?



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos de Expressões

- **Literais**
  - Expressão mais simples

```
99.0      99        0143      'c'       0x43
```

- **Agregação**
  - Constrói um valor a partir de seus componentes:

```
int c[] = {1, 2, 3};
struct data d = {1, 7, 1999};
char *x = {'a', 'b', 'c', '\0'};
int b[6] = {0};
char *y = "abc";
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos de Expressões

- **Expressões Aritméticas**
  - Operadores aritméticos: *+ - * / % etc…*
- **Expressões Relacionais**
  - Operadores relacionais: *>, <, == etc…*
- **Expressões Booleanas**
  - Operadores booleanos: *and, or, etc..*
- **Expressões Condicionais**
  - Operadores condicionais: *"**? :**" em C or "**if else**" em Python*
  - Exemplos: expressão_1 **?** expressão_2 **:** expressão_3

    (count == 0) **?** 0 **:** sum / count

    (count == 0) **if** 0 **else** sum / count

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos de Expressões

- **Expressões Binárias**
  - Operadores binários:

```
int main() {
     int j = 10;
     char c = 2;
     printf("%d\n", ~0);       /* imprime -1 */
     printf("%d\n", j & c);    /* imprime  2 */
     printf("%d\n", j | c);    /* imprime 10 */
     printf("%d\n", j ^ c);    /* imprime  8 */
     printf("%d\n", j << c);   /* imprime 40 */
     printf("%d\n", j >> c);   /* imprime  2 */
 }
```

- **Chamadas de função**
  - Operadores: *nome da função*
  - Operandos: *parâmetros*
  - *Exemplo: Chamada condicional de função em ML:*

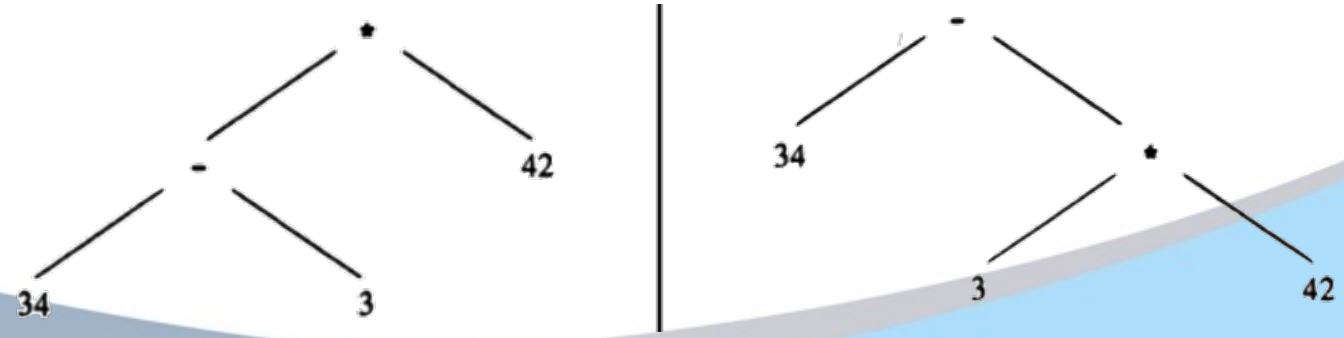
```
val taxa =
  (if difPgVen > 0 then desconto else multa)(difPgVen)
```

Precedência de operadores

- Definidas por regras de precedência

	Ruby	*C-Based Languages*
Highest	**	postfix ++, --
	unary +, -	prefix ++, --, unary +, -
	*, /, %	*, /, %
Lowest	binary +, -	binary +, -

- Muito importante para evitar ambiguidades, i.e. mais de uma forma de interpretar o mesmo código



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Precedência de operadores

- Ambiguidades causam problemas sérios para os compiladores.
- Algumas LPs não definem precedência (Smalltalk e APL);
- Influenciam na redigibilidade/legibilidade:
  - Ausência ou escolha inadequada de precedência pode ser prejudicial.
  - Parênteses asseguram a ordem, mas uso também pode ser prejudicial e inclusive impedir otimizações.
  - Use parênteses para evitar dúvidas, mas sem excesso!

```
(* Erro em Pascal: and tem precedência sobre > e < *)
if a > 5 and b < 10 then

(* OK. Precedência ajustada com parênteses. *)
if (a > 5) and (b < 10) then
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Precedência de operadores

- Operadores aritméticos, relacionais e booleanos em LPs baseadas em C:

Mais alta	++ e – posfixados
	+ unário, – unário, ++, –– e ! prefixados
	*, /, %
	+ binário, – binário
	<, >, <=, >=
	=, !=
	&&
Mais baixa	||

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Associatividade de operadores

- Se operadores tem a mesma precedência (ou se não há regras de precedência), utiliza-se regras de associatividade
  - à esquerda (left associativity)
  - à direita (right associativity)

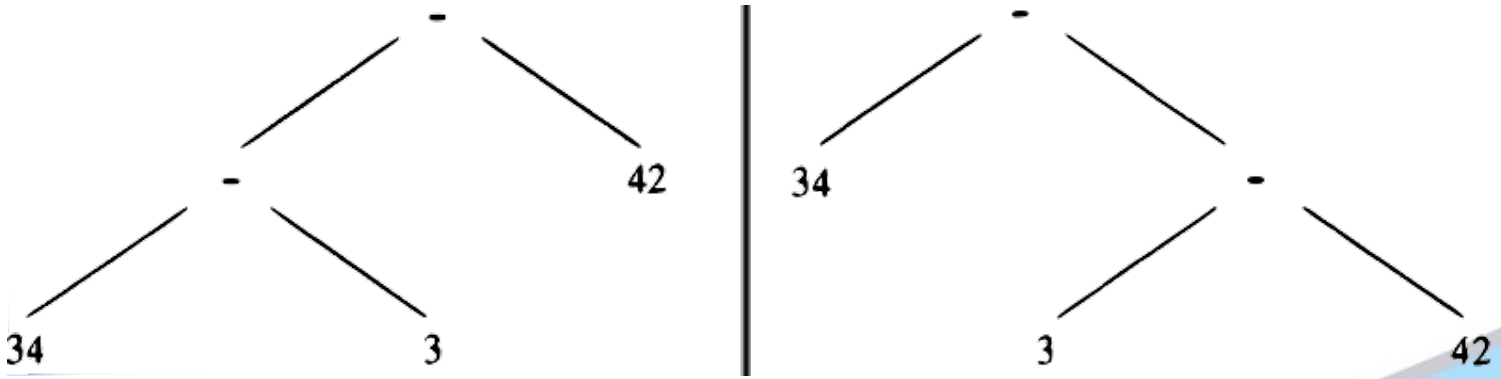
Language	*Associativity Rule*
Ruby	Left: *, /, +, - Right: **
C-based languages	Left: *, /, %, binary +, binary - Right: ++, --, unary -, unary +

- A maioria dos operadores são associativos à esquerda.
  - Há exceções:

```
x = **p;            // *(*p)
if (!!x) y = 3;     // !(!x)
a = b = c;          // a = (b = c);
```

Associatividade de operadores

- Na matemática não faz diferença, em programação sim
- Um mesmo operador ainda pode ser ambíguo!
- Duas ASTs para a mesma entrada: 34 - 3 - 42



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Ordem de avaliação dos operandos

- Por que associatividade de LPs não é igual à matemática?
  - *Overflow* / *underflow* (transbordamento [negativo])

```
// Suponha f() e h() retornam números positivos grandes
// enquanto g() retorna um número negativo grande.
x = f() + g() + h();
```

  - Efeitos colaterais (*side effects*)

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Efeitos colaterais

- Efeitos colaterais de uma função ou expressão
  - Modificar um dos argumentos
  - Modificar uma variável global
  - Modificar o mundo real (I/O)
    - printf é chamada exclusivamente pelo seu efeito colateral

```c
int fun(int *x) {
    *x = 20;
    return 10;
}

int main() {
    int a, b;

    a = 10;
    b = a + fun(&a);
    printf("%d %d\n", a, b); // 20 30

    a = 10;
    b = a;
    b += fun(&a);
    printf("%d %d\n", a, b); // 20 20
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Efeitos colaterais

- Funções na matemática não têm efeitos colaterais
  - Funções são somente mapeamentos
  - Não existe o conceito de variável como em LPs
- O mesmo vale para linguagens funcionais puras (Haskell, etc)

```
λ let x = 4 in x * x
16
:: Num a => a
λ x
Not in scope: ‘x’
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Efeitos colaterais

- Em LPs funcionais variáveis são *aliases*/apelidos para expressões
- Variáveis só podem ser atribuídas uma única vez
  - ⇒ Programas em LP funcionais não têm estado!
  - Não é totalmente verdade: estado "emerge" das chamadas de funções
- LPs imperativas manipulam memória ⇒ estado
- *State* vs. *stateless* é a grande discussão atual em LPs
- Proponentes de LPs funcionais indicam como vantagens:
  - Facilidade de depuração/paralelização entre outras
- Ainda está em aberto, mas FP vem ganhando espaço
- Efeitos colaterais sempre vão existir ⇒ I/O!

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Transparência Referencial

- Um programa tem a propriedade de transparência referencial se quaisquer duas expressões que tem o mesmo valor puderem ser substituídas uma pela outra sem afetar o resultado.

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

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Sobrecarga de operadores

- Múltiplo uso de um operador
  - Exemplo: em algumas LPS, o sinal + pode ser usado para adição de números inteiros, “reais” ou ainda para concatenação de listas

- É aceitável, desde que não prejudique nem legibilidade nem confiabilidade.

- Em algumas LPs a sobrecarga também pode ser definida pelo programador. Alguns operadores podem não ser sobrecarregáveis. Outras LPs como Java não permitem tal sobrecarga.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Conversões de tipos

- Estreitamento (*narrowing*)
  - Converte um tipo "maior" em um tipo "menor"
  - double -> float
  - Requer uma indicação explícita ⇒ perda de precisão
- Alargamento / ampliação (*widening*)
  - Converte um tipo "menor" em um tipo "maior"
  - int -> float
  - Geralmente ocorre implicitamente
- Expressões de modo misto (*mixed-mode*)
  - Operandos com tipos diferentes
  - Exigem conversão implícita de tipos

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Conversões de tipos

- Conversão implícita de tipos em uma LP com tipagem estática

Type of			Code
a	b	a + b	
integer	integer	integer	iADD r_a, r_b ⇒ r_{a+b}
integer	real	real	i2f f_a ⇒ r_{a_f} fADD r_{a_f}, r_b ⇒ r_{a_f+b}
integer	double	double	i2d r_a ⇒ r_{a_d} dADD r_{a_d}, r_b ⇒ r_{a_d+b}
real	real	real	fADD r_a, r_b ⇒ r_{a+b}
real	double	double	r2d r_a ⇒ r_{a_d} dADD r_{a_d}, r_b ⇒ r_{a_d+b}
double	double	double	dADD r_a, r_b ⇒ r_{a+b}

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Curto-circuito

- Elimina avaliação de algum operando
- Não é razoável se fazer em expressões aritméticas
- Simples de fazer em expressões Booleanas
- Facilitam escrita de código em alguns casos (abaixo)
- Podem causar bugs sutis quando há efeitos colaterais

```
index = 0;
while ((index < listlen) && (list[index] != key))
  index = index + 1;
```

Sentenças de Atribuição

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Sentenças de Atribuição

- Comando ou efeito colateral?
- Um comando de **atribuição** modifica uma área de memória
  - ⇒ Efeito colateral
  - LPs imperativas focam em modificações de variáveis
  - ⇒ LPs imperativas vivem de efeitos colaterais

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

- Operador de Atribuição Composta:

```
sum += value;
```

- Atribuição Múltipla:

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

- Atribuição como Expressão:

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