

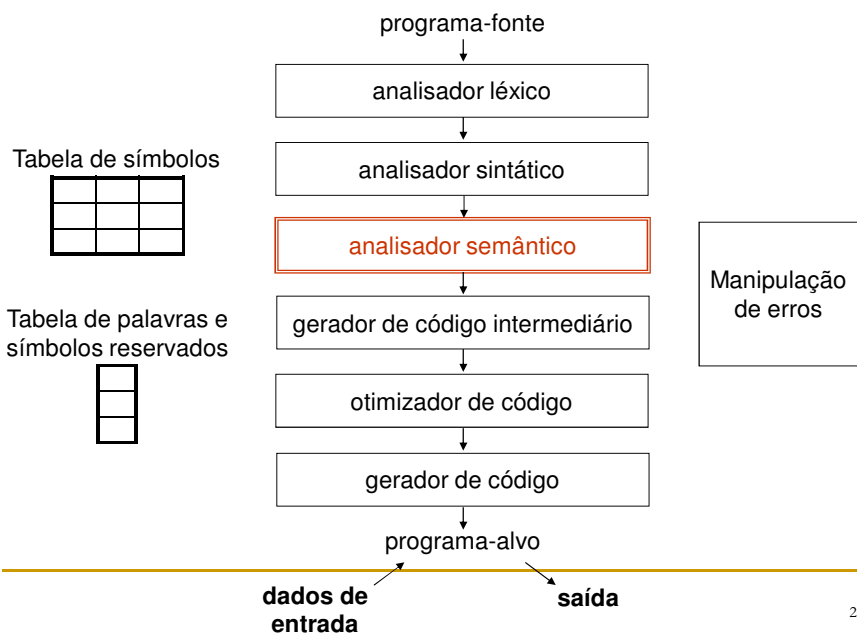
# Análise semântica (continuação)

Função, interação com o compilador  
Tabela de símbolos  
Análise semântica

Prof. Thiago A. S. Pardo

1

## Estrutura geral de um compilador



2

## Gramática de atributos

- Gramática de atributos
  - Método usualmente utilizado para formalização semântica
  - Conjunto de **atributos** e **regras semânticas** para uma gramática
    - Cada regra sintática/gramatical pode ter regras semânticas associadas
  - **Atributos** associados aos símbolos gramaticais
    - Por exemplo, valor e escopo
      - x.valor, x.escopo
  - **Regras semânticas** que manipulam os atributos
    - Por exemplo, regra para somar os atributos valores de duas variáveis
      - $x:=a+b$ , cuja regra é  $x.valor:=a.valor+b.valor$

3

## Gramática de atributos

- Considerando a gramática abaixo

número  $\rightarrow$  num sufixo

sufixo  $\rightarrow$  b | d

num  $\rightarrow$  num dígito | dígito

dígito  $\rightarrow$  0|1|2|3|4|5|6|7|8|9

4

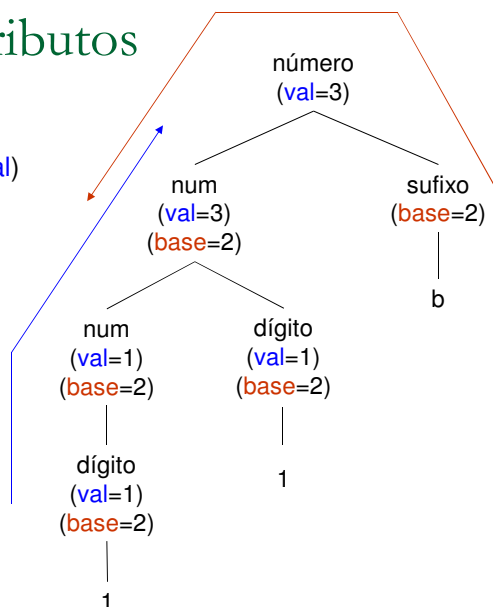
## Gramática de atributos

Regras gramaticais	Regras semânticas
número → num sufixo	número.val = num.val num.base = sufixo.base
sufixo → b	sufixo.base = 2
sufixo → d	sufixo.base = 10
num <sub>1</sub> → num <sub>2</sub> dígito	num <sub>1</sub> .val = <b>if</b> dígito.val = erro <b>or</b> num <sub>2</sub> .val=erro <b>then</b> erro <b>else</b> num <sub>2</sub> .val * num <sub>1</sub> .base + dígito.val num <sub>2</sub> .base = num <sub>1</sub> .base dígito.base = num <sub>1</sub> .base
num → dígito	num.val = dígito.val dígito.base = num.base
dígito → 0	dígito.val = 0
dígito → 1	dígito.val = 1
dígito → 2	dígito.val = <b>if</b> dígito.base=2 <b>then</b> erro <b>else</b> 2
...	...

## Gramática de atributos

- Atenção
  - Alguns valores sobem (val)
  - Outros descem (base)

Como calcular os atributos de forma consistente?



## Cômputo de atributos

- Com base na árvore sintática explícita
  - Grafos de dependência
    - Compilador de mais de uma passagem
  
- *Ad hoc*
  - Análise semântica “comandada” pela análise sintática
    - Compilador de uma única passagem

7

## Cômputo de atributos

- Grafos de dependência
  - Especificam a ordem de cômputo dos atributos de cada regra gramatical em uma árvore sintática
    - Portanto, um grafo associado a cada regra gramatical
  
  - Para uma cadeia da linguagem, tem-se um grafo composto por todos os subgrafos

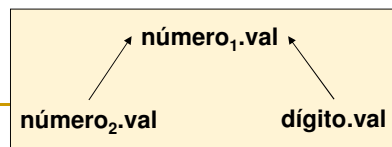
8

## Cômputo de atributos

- Exemplo

número  $\rightarrow$  número dígito | dígito  
 dígito  $\rightarrow$  0|1|2|3|4|5|6|7|8|9

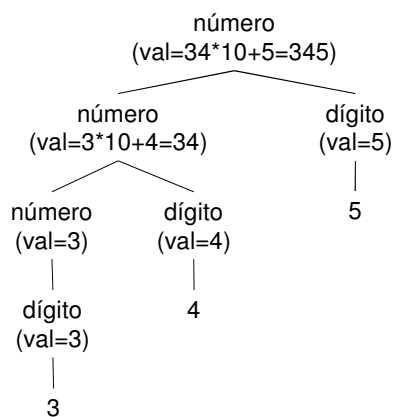
Regras gramaticais	Regras semânticas
número <sub>1</sub> $\rightarrow$ número <sub>2</sub> dígito	número <sub>1</sub> .val $\rightarrow$ número <sub>2</sub> .val * 10 + dígito.val
número $\rightarrow$ dígito	Número.val = dígito.val
dígito $\rightarrow$ 0	dígito.val = 0
dígito $\rightarrow$ 1	dígito.val = 1
...	...
dígito $\rightarrow$ 9	dígito.val = 2



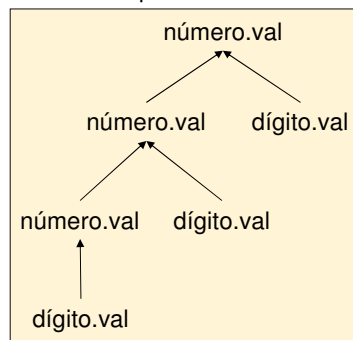
9

## Cômputo de atributos

- Cadeia 345



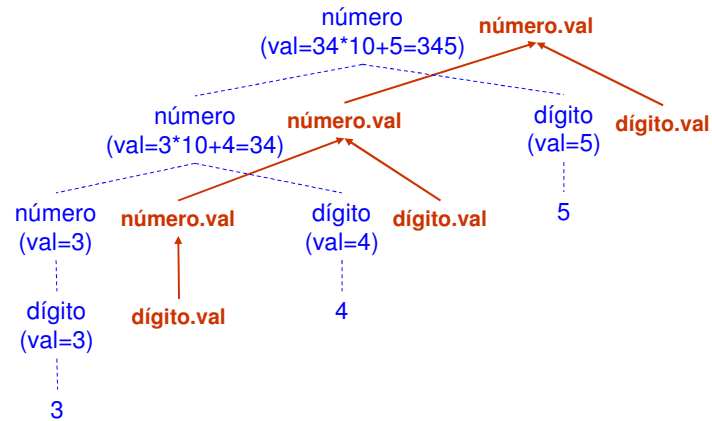
Grafo de dependência



10

## Cômputo de atributos

- Grafo amarrado à árvore sintática



11

## Cômputo de atributos

- Considere a gramática abaixo e sua gramática de atributos

decl  $\rightarrow$  tipo var-lista

tipo  $\rightarrow$  int | float

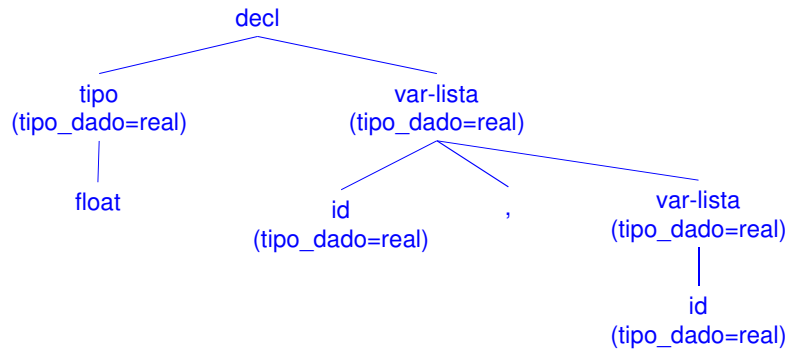
var-lista  $\rightarrow$  id, var-lista | id

Regras gramaticais	Regras semânticas
decl $\rightarrow$ tipo var-lista	var-lista.tipo_dado = tipo.tipo_dado
tipo $\rightarrow$ int	tipo.tipo_dado = integer
tipo $\rightarrow$ float	tipo.tipo_dado = real
var-lista <sub>1</sub> $\rightarrow$ id, var-lista <sub>2</sub>	id.tipo_dado = var-lista <sub>1</sub> .tipo_dado var-lista <sub>2</sub> .tipo_dado = var-lista <sub>1</sub> .tipo_dado
var-lista $\rightarrow$ id	id.tipo_dado=var-lista.tipo_dado

12

## Cômputo de atributos

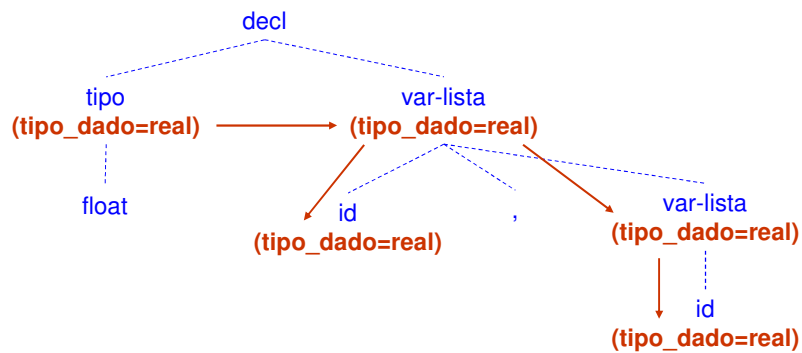
- Para a cadeia float x,y



13

## Cômputo de atributos

- Para a cadeia float x,y



14

## Cômputo de atributos

- **Exercício:** considere a gramática abaixo e sua gramática de atributos

número  $\rightarrow$  num sufixo  
sufixo  $\rightarrow$  b | d  
num  $\rightarrow$  num dígito | dígito  
dígito  $\rightarrow$  0|1|2|3|4|5|6|7|8|9

15

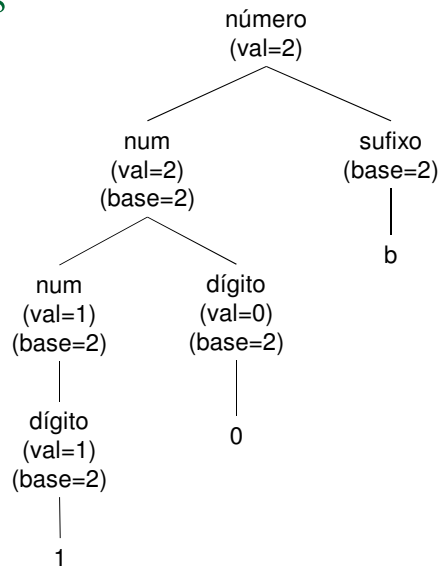
## Cômputo de atributos

Regras gramaticais	Regras semânticas
número $\rightarrow$ num sufixo	número.val = num.val num.base = sufixo.base
sufixo $\rightarrow$ b	sufixo.base = 2
sufixo $\rightarrow$ d	sufixo.base = 10
num <sub>1</sub> $\rightarrow$ num <sub>2</sub> dígito	num <sub>1</sub> .val = <b>if</b> dígito.val = erro <b>or</b> num <sub>2</sub> .val=erro <b>then</b> erro <b>else</b> num <sub>2</sub> .val * num <sub>1</sub> .base + dígito.val num <sub>2</sub> .base = num <sub>1</sub> .base dígito.base = num <sub>1</sub> .base
num $\rightarrow$ dígito	num.val = dígito.val dígito.base = num.base
dígito $\rightarrow$ 0	dígito.val = 0
dígito $\rightarrow$ 1	dígito.val = 1
dígito $\rightarrow$ 2	dígito.val = <b>if</b> dígito.base=2 <b>then</b> erro <b>else</b> 2
...	...



## Cômputo de atributos

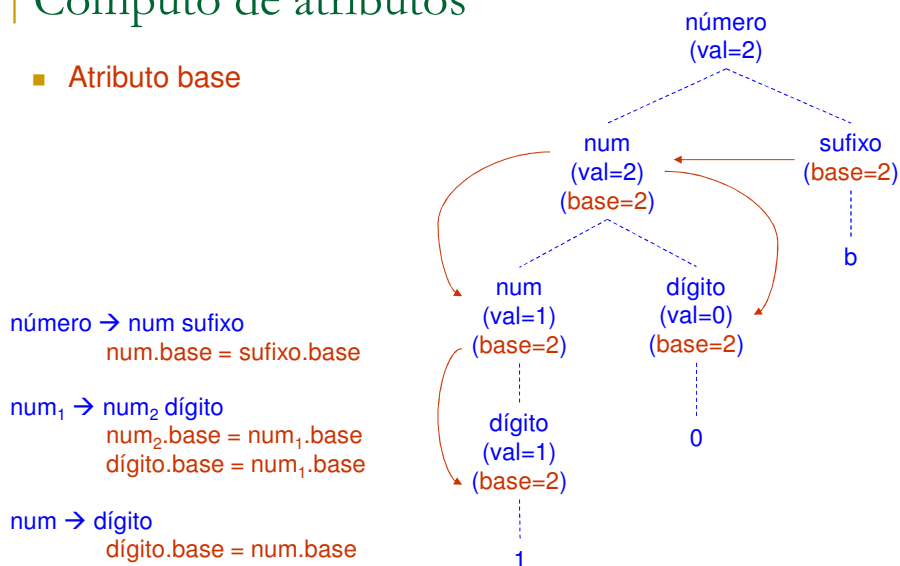
- Dada a árvore sintática da cadeia 10b, construa o grafo de dependência



17

## Cômputo de atributos

- Atributo base



18

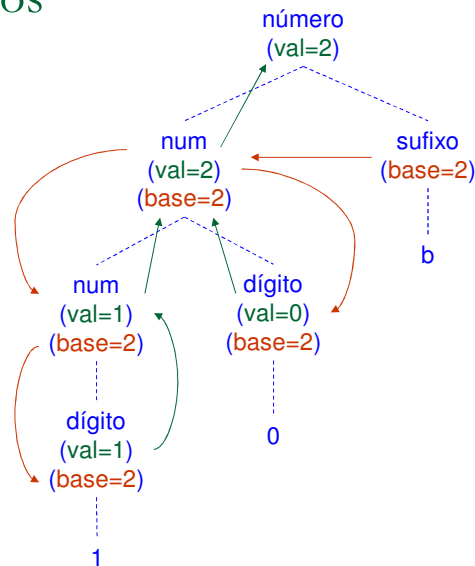
## Cômputo de atributos

- Atributo base e atributo valor

número → num sufixo  
 $\text{número.val} = \text{num.val}$

$\text{num}_1 \rightarrow \text{num}_2 \text{ dígito}$   
 $\text{num}_1.\text{val} = \text{num}_2.\text{val} * \text{num}_1.\text{base} + \text{dígito.val}$

$\text{num} \rightarrow \text{dígito}$   
 $\text{num.val} = \text{dígito.val}$

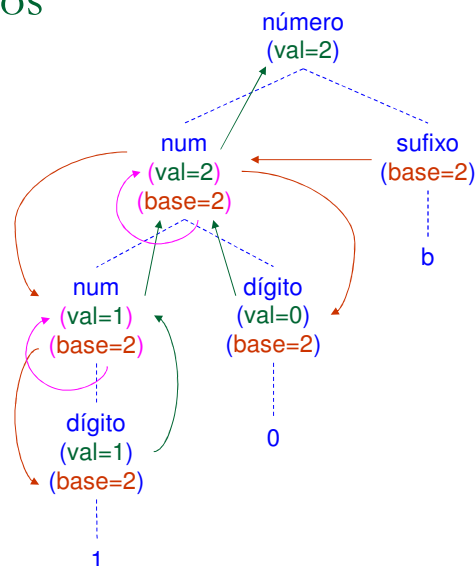


19

## Cômputo de atributos

- Atributo base e atributo valor e dependência do valor em relação à base

$\text{num}_1 \rightarrow \text{num}_2 \text{ dígito}$   
 $\text{num}_1.\text{val} =$   
 if  $\text{dígito.val} = \text{erro}$  or  $\text{num}_2.\text{val} = \text{erro}$  then erro  
 else  $\text{num}_2.\text{val} * \text{num}_1.\text{base} + \text{dígito.val}$



20

## Cômputo de atributos

### ■ Dois tipos de atributos

- **Atributos herdados**: dependências de pais para filhos e/ou entre irmãos (p.ex., atributo *base* da gramática anterior)
- **Atributos sintetizados**: dependências apontam de filhos para pais (p.ex., atributo *val* da gramática anterior)
  - Uma gramática que só tem atributos sintetizados é denominada **gramática S-atribuída**

21

## Cômputo de atributos

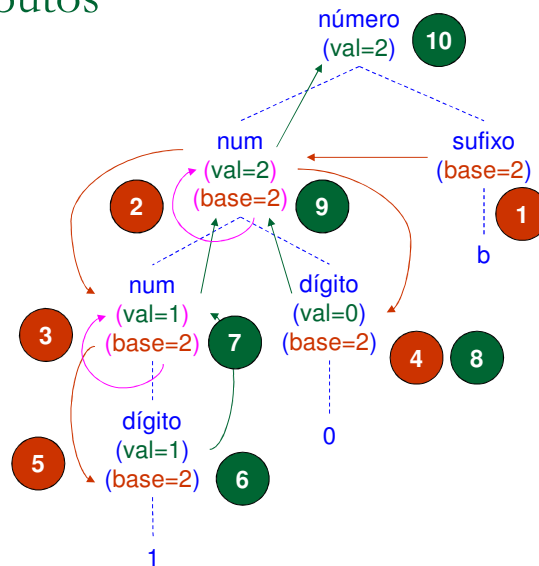
### ■ Ordem de cômputo dos atributos

- Os atributos que não dependem de outros atributos devem ser computados primeiro, logicamente
- Opções
  - **Ordenação topológica** do grafo de dependências: método de árvore de análise sintática
  - **Manualmente**, determinado pelo projetista do compilador: método baseado em regras

22

## Cômputo de atributos

- **Ordenação topológica**
  - Relembrando ALG2
- Em geral, inicia-se pelos atributos independentes dos nós folha
- Dificuldades do método
  - Construção completa do grafo de dependência
  - Grafos acíclicos para gramáticas de atributos acíclicas (um algoritmo para tal verificação é exponencial em tempo)



23

## Cômputo de atributos

- **Método baseado em regras**
  - Adotado em praticamente todos os compiladores
  - O projetista do compilador analisa a gramática de atributos e seus grafos de dependência e determina a ordem de cômputo dos atributos
    - Em geral, não é muito complicado de se fazer
  - Para cada regra gramatical, definição do percurso realizado no trecho correspondente na árvore sintática
    - Possibilidades: percurso em-ordem, pré-ordem, pós-ordem ou arbitrário

24

## Cômputo de atributos

- Exemplos de percursos para uma árvore binária (relembrando [ALG1](#))
  - Percurso em-ordem
    - Filho esquerdo, raiz (nó pai, isto é, lado esquerdo da regra em foco), filho direito
  - Percurso pré-ordem
    - Raiz, filho esquerdo, filho direito
  - Percurso pós-ordem
    - Filho esquerdo, filho direito, raiz

25

## Cômputo de atributos

- Exemplo: sub-rotina para computar o atributo `tipo_dado` da gramática abaixo

`decl` → `tipo var-lista`

`tipo` → **int** | **float**

`var-lista` → **id**, `var-lista` | **id**

Regras gramaticais	Regras semânticas
<code>decl</code> → <code>tipo var-lista</code>	<code>var-lista.tipo_dado = tipo.tipo_dado</code>
<code>tipo</code> → <code>int</code>	<code>tipo.tipo_dado = integer</code>
<code>tipo</code> → <code>float</code>	<code>tipo.tipo_dado = real</code>
<code>var-lista<sub>1</sub></code> → <code>id, var-lista<sub>2</sub></code>	<code>id.tipo_dado = var-lista<sub>1</sub>.tipo_dado</code> <code>var-lista<sub>2</sub>.tipo_dado = var-lista<sub>1</sub>.tipo_dado</code>
<code>var-lista</code> → <code>id</code>	<code>id.tipo_dado=var-lista.tipo_dado</code>

26

## Cômputo de atributos

- Exemplo: sub-rotina para computar o atributo `tipo_dado` da gramática abaixo

```
procedure AvalTipo(T: nó_árvore);
```

```
begin
```

```
  case nó de T of
```

```
    decl:
```

```
      AvalTipo(tipo);
```

```
      var-lista.tipo_dado:=tipo.tipo_dado;
```

```
      AvalTipo(var-lista);
```

```
    tipo:
```

```
      if filho de T=int then T.tipo_dado:=inteiro
```

```
      else T.tipo_dado:=real;
```

```
    var-lista:
```

```
      atribui T.tipo_dado a primeiro filho de T
```

```
      if terceiro filho de T não é NIL then
```

```
        atribui T.tipo_dado a terceiro filho;
```

```
        AvalTipo(terceiro filho de T);
```

```
  end;
```

```
end;
```

pré-ordem

decl → tipo var-lista

tipo → int | float

var-lista → id, var-lista | id

## Cômputo de atributos

- Opcionalmente, **valores de atributos** podem ser associados a **parâmetros** ou **valores de retorno** de sub-rotinas de cômputo de atributos em vez de serem armazenados nos nós de uma árvore sintática
  - Interessante para a situação em que muitos atributos são usados apenas temporariamente ou como suporte para cômputo de outros atributos
  - Normalmente, **atributos herdados são passados via parâmetros** e **atributos sintetizados via valor de retorno**

## Cômputo de atributos

### ■ Exemplo

número → num sufixo  
sufixo → b | d  
num → num dígito | dígito  
dígito → 0|1|2|3|4|5|6|7|8|9

29

## Cômputo de atributos

Regras gramaticais	Regras semânticas
número → num sufixo	número.val = num.val num.base = sufixo.base
sufixo → b	sufixo.base = 2
sufixo → d	sufixo.base = 10
num <sub>1</sub> → num <sub>2</sub> dígito	num <sub>1</sub> .val = <b>if</b> dígito.val = erro <b>or</b> num <sub>2</sub> .val=erro <b>then</b> erro <b>else</b> num <sub>2</sub> .val * num <sub>1</sub> .base + dígito.val num <sub>2</sub> .base = num <sub>1</sub> .base dígito.base = num <sub>1</sub> .base
num → dígito	num.val = dígito.val dígito.base = num.base
dígito → 0	dígito.val = 0
dígito → 1	dígito.val = 1
dígito → 2	dígito.val = <b>if</b> dígito.base=2 <b>then</b> erro <b>else</b> 2
...	...

```

function AvalComBase(T: nó_árvore; base: inteiro): inteiro;
var temp, temp2: inteiro;
begin
  case nó de T of
    número:
      temp:=AvalComBase(filho à direita de T,0);
      return AvalComBase(filho à esquerda de T,temp);
    num:
      temp:=AvalComBase(filho à esquerda de T,base);
      if filho à direita de T não é NIL then
        temp2:=AvalComBase(filho à direita de T,base);
        if temp<>erro and temp2<>erro then
          return base*temp+temp2
        else return erro;
      else return temp;
    sufixo:
      if filho de T=b then return 2
      else return 10;
    dígito:
      if base=2 and filho de T>1 then return erro
      else return numval(filho de T);
  end;
end;

```

número → num sufixo  
sufixo → b | d  
num → num dígito | dígito  
dígito → 0|1|2|3|4|5|6|7|8|9

## Cômputo de atributos

- Com base na árvore sintática explícita
  - Grafos de dependência
    - Compilador de mais de uma passagem
  
- *Ad hoc*
  - **Análise semântica “comandada” pela análise sintática**
    - Compilador de uma única passagem



## Cômputo de atributos

- Estruturas de dados externas
  - Em vez de se armazenar os atributos na árvore sintática ou de manipulá-los via parâmetros e valores de retornos, os atributos podem ser armazenados em estruturas separadas
    - Variáveis globais
    - Listas
    - Tabelas
  - Em compilação, a **tabela de símbolos** é utilizada, em geral

33

## Tabela de símbolos

- Estrutura principal da compilação
- Captura a **sensibilidade ao contexto** e as ações executadas no decorrer do programa
- Pode estar atrelada a todas as etapas da compilação
- Permite a realização da análise semântica
- Fundamental na geração de código

34

## Tabela de símbolos

- Permite saber durante a compilação de um programa o **tipo** e o **valor** de seus **elementos** (números e identificadores), **escopo** destes, número e tipo dos **parâmetros** de um procedimento, etc.
  - Cada token tem atributos/informações diferentes associadas

Cadeia	Token	Categoria	Tipo	Valor	...
i	id	var	integer	1	...
fat	id	proc	-	-	...
2	num	-	integer	2	...
...					

35

## Tabela de símbolos

- Exemplo de atributos de identificador de **variável**
  - Tipo de variável (inteira, real, etc.), nome da variável, endereço na memória, escopo (programa principal, função, etc.), etc.
- Para **vetor**, ainda seriam necessários atributos de tamanho do vetor, o valor de seus limites, etc.

36

## Tabela de símbolos

- Principais operações efetuadas
  - **Inserir**: armazena na tabela informações fornecidas pelas declarações no programa
  - **Busca**: recupera da tabela informações de um elemento declarado no programa quando esse elemento é utilizado
  - **Remover**: remove (ou torna inacessível) da tabela informações sobre um elemento declarado que não se mostra mais necessário no programa
- As especificidades dessas operações são **dependentes da linguagem de programação** em questão

37

## Tabela de símbolos

- A **tabela é acessada** pelo compilador sempre que um elemento é mencionado no programa
  - Verificar ou incluir sua declaração
  - Verificar seu tipo, escopo ou alguma outra informação
  - Atualizar alguma informação associada ao identificador (por exemplo, valor)
  - Remover um elemento quando este não se faz mais necessário ao programa

38

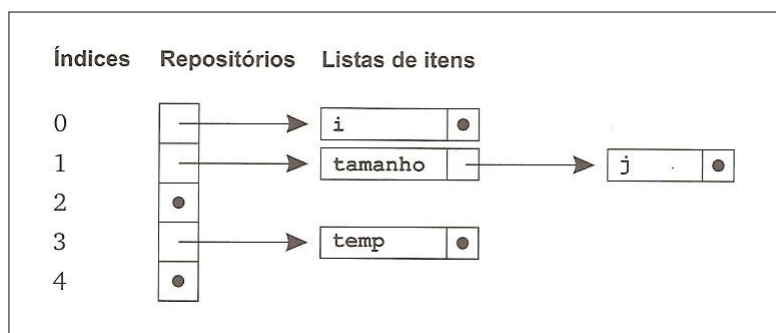
## Tabela de símbolos

- **Estrutura** da tabela de símbolos: determinada pela eficiência das operações de inserir, verificar e remover
- Várias possibilidades
  - Implementação
    - Estática
    - Dinâmica: melhor opção
  - Estrutura
    - Listas, matrizes
    - Árvores de busca (por exemplo, B e AVL)
  - Acesso
    - Sequencial, busca binária, etc.
    - *Hashing*: opção mais eficiente
      - O elemento do programa é a chave e a função *hash* indica sua posição na tabela de símbolos
      - Necessidade de tratamento de colisões

39

## Tabela de símbolos

- Exemplo de *hashing* com resolução de colisões para a inclusão dos identificadores *i*, *j*, *tamanho* e *temp*



40

## Tabela de símbolos

- Questões de projeto
  - Tamanho da tabela: tipicamente, de algumas centenas a mil “linhas”
    - Dependente da forma de implementação
      - Na implementação dinâmica, não é necessário se preocupar tanto com isso
  - Uma única tabela para todas as declarações ou várias tabelas, sendo uma para cada tipo de declaração (constantes, variáveis, tipos, procedimentos e funções)
    - Por quê?

41

## Tabela de símbolos

- Questões de projeto
  - Tamanho da tabela: tipicamente, de algumas centenas a mil “linhas”
    - Dependente da forma de implementação
      - Na implementação dinâmica, não é necessário se preocupar tanto com isso
  - Uma única tabela para todas as declarações ou várias tabelas, sendo uma para cada tipo de declaração (constantes, variáveis, tipos, procedimentos e funções)
    - Diferentes declarações têm diferentes informações/atributos (por exemplo, variáveis não têm número de argumentos, enquanto procedimentos têm)

42

## Tabela de símbolos

- Representação de **escopo** de identificadores do programa: várias tabelas ou uma única tabela com a identificação do escopo (como um atributo ou por meio de listas ligadas, por exemplo) para cada identificador
  - Tratamento de escopo
    - Inserção de identificadores de mesmo nome, mas em níveis diferentes
    - Remoção de identificadores cujos escopos deixaram de existir
  - Em geral, na maioria das linguagens de programação, aplica-se a “regra do aninhamento mais próximo”

43

## Tabela de símbolos

- Possibilidades para tratamento de escopos
  - Inclusão de um **campo a mais** na tabela de símbolos indicando o nível da variável no programa
    - Controle do nível durante a compilação do programa
      - Quando se chama um procedimento (ou função), faz-se **nível=nível+1**
      - Quando se sai de um procedimento (ou função), faz-se **nível=nível-1**
  - Associação das variáveis locais a um procedimento (ou função) à entrada da tabela para o procedimento (ou função) por meio, por exemplo, de uma **lista encadeada**
    - Atenção: para a checagem de tipos, deve-se saber quantos são e quais são os parâmetros de um procedimento (ou função) na tabela de símbolos
  - **Tabelas diferentes** para diferentes escopos

44

## Tabela de símbolos

- **Tratamento de escopo**

- Como diferenciar variáveis globais de locais
  - Tratamento de variáveis de mesmo nome, mas de escopos diferentes

```
program meu_prog
  procedure meu_proc(x: integer)
    var y: real
    begin
      read(y);
      x:=x+y
    end;
  var x, y: integer
  begin
    read(y);
    x:=x*y
  end.
```

45

## Exercício

- Gere a tabela de símbolos para o programa abaixo

```
program meu_prog
  procedure meu_proc(x: integer)
    var y: real
    begin
      read(y);
      x:=x+y
    end;
  var x, y: integer
  begin
    read(y);
    x:=x*y
  end.
```

46

## Tabela de símbolos

- Exemplo de tratamento de escopo
  - Sub-rotinas aninhadas
  - Variáveis globais e locais com mesmo nome

```

program Ex;
var i,j: integer;

function f(tamanho: integer): integer;
var i,temp: char;

    procedure g;
    var j: real;
    begin
        ...
    end;

    procedure h;
    var j: ^char;
    begin
        ...
    end;

begin (* f *)
    ...
end;

begin (* programa principal *)
    ...
end.
    
```

## Tabela de símbolos

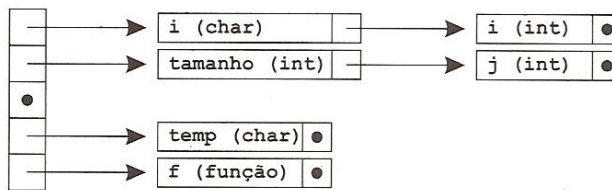
- Exemplo de tratamento de escopo
  - Sub-rotinas aninhadas
  - Variáveis globais e locais com mesmo nome

```

program Ex;
var i,j: integer;

function f(tamanho: integer): integer;
var i,temp: char;
    
```

Repositórios    Listas de itens



(a) Após o processamento das declarações do corpo de f

```

end;

begin (* f *)
    ...
end;

begin (* programa principal *)
    ...
end.
    
```



## Tabela de símbolos

- Exemplo de tratamento de escopo
  - Sub-rotinas aninhadas
  - Variáveis globais

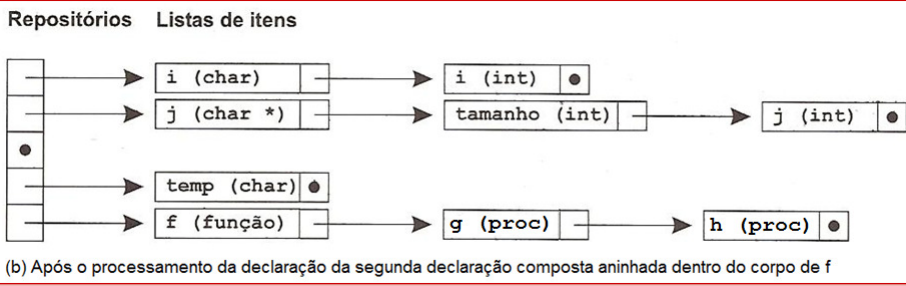
```

program Ex;
var i,j: integer;

function f(tamanho: integer): integer;
var i,temp: char;

    procedure g;
    var j: real;
    begin
        ...
    end;

    procedure h;
    var j: ^char;
    
```



## Tabela de símbolos

- Exemplo de tratamento de escopo
  - Sub-rotinas aninhadas
  - Variáveis globais

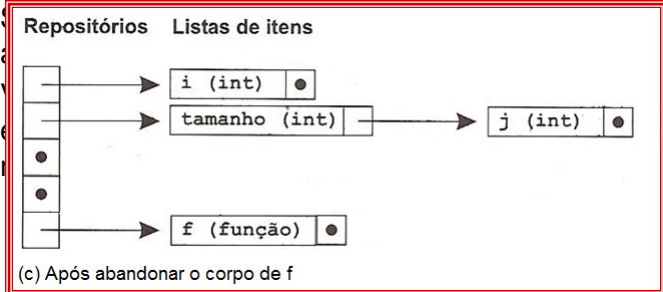
```

program Ex;
var i,j: integer;

function f(tamanho: integer): integer;
var i,temp: char;

    procedure g;
    var j: real;
    begin
        ...
    end;

    procedure h;
    var j: ^char;
    
```



```

begin (* programa principal *)
    ...
end.
    
```

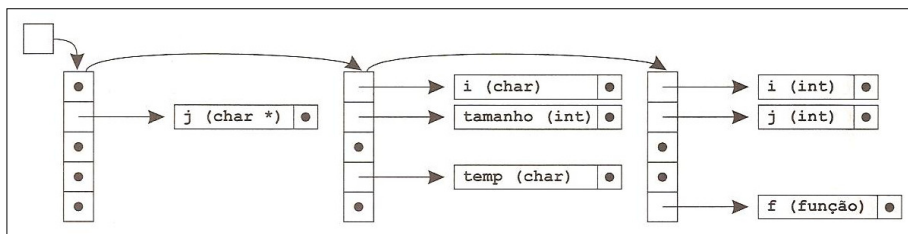
## Tabela de símbolos

- Comportamento de **pilha**
  - Insere as declarações mais recentes, ocultando as antigas
  - Remove as mais recentes, voltando ao escopo anterior
  - Acesso às mais recentes

51

## Tabela de símbolos

- Alternativa para tabela anterior: tabelas separadas para cada escopo
  - Mudar o escopo requer apenas a mudança do ponteiro



52

## Tabela de símbolos

- A tabela de símbolos pode ser utilizada para armazenar as palavras reservadas e símbolos especiais da linguagem, podendo dispensar o uso da tabela de palavras e símbolos reservados

53

## Tabela de símbolos

- **Descritores**
  - Registros (campos) que formam a tabela de símbolos
  - Armazenam as informações dos números e identificadores
- Diferentes identificadores têm diferentes descritores
  - Tem que se levar isso em consideração no projeto da tabela de símbolos para sua otimização
  - Possibilidade de se usar uniões para o caso de uma mesma tabela para tudo

54

## Tabela de símbolos

- **Inserção** de elementos na tabela
  - Associação de regras semânticas às regras gramaticais
  - Verificar se o elemento já não consta na tabela
  
- **Busca** de informação na tabela
  - Realizada antes da inserção
  - Busca de informações para análise semântica
  
- **Remoção** de elementos da tabela
  - Tornar inacessíveis dados que não são mais necessários (por exemplo, após o escopo ter terminado)
  - Linguagens que permitem estruturação em blocos

55

## Tabela de símbolos

- As sub-rotinas de inserção, busca e remoção podem ser inseridas diretamente na **gramática de atributos**
  - Explicitamente, via chamadas de sub-rotinas de manipulação da tabela
    - Compilação em mais de uma passagem, se árvore sintática é a base para a análise
    - Opcionalmente, compilação de uma única passagem, sendo a gramática de atributos utilizada apenas para a descrição da semântica

56

## Tabela de símbolos

- Exemplo: decl → tipo var-lista  
tipo → **int** | **float**  
var-lista → **id**, var-lista | **id**

Regras gramaticais	Regras semânticas
decl → tipo var-lista	var-lista.tipo_dado = tipo.tipo_dado
tipo → int	tipo.tipo_dado = integer
tipo → float	tipo.tipo_dado = real
var-lista <sub>1</sub> → id, var-lista <sub>2</sub>	id.tipo_dado = var-lista <sub>1</sub> .tipo_dado var-lista <sub>2</sub> .tipo_dado = var-lista <sub>1</sub> .tipo_dado If busca(id)=FALSE then inserir(id,id.tipo_dado) else ERRO("identificador já declarado")
var-lista → id	id.tipo_dado=var-lista.tipo_dado If busca(id)=FALSE then inserir(id,id.tipo_dado) else ERRO("identificador já declarado")

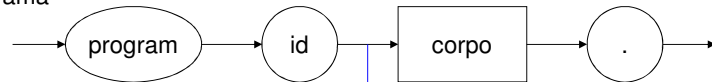
## Tabela de símbolos

- As sub-rotinas de inserção, busca e remoção podem ser **inseridas diretamente na análise sintática**
  - Solução *ad hoc*
    - Compilação de uma única passagem

## Tabela de símbolos

- **Inserção** de elementos na tabela
  - Declaração, principalmente

programa



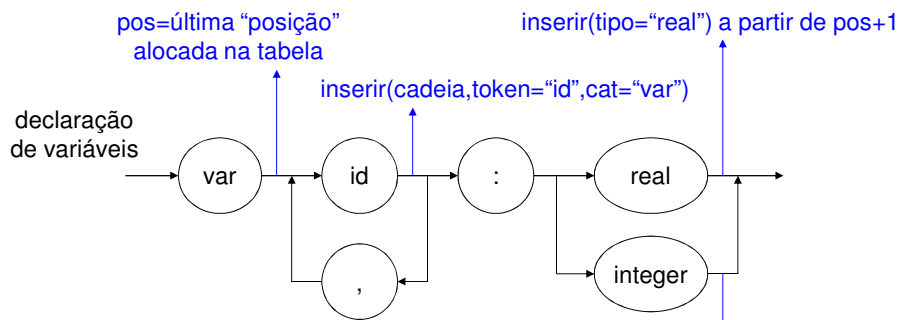
inserir(cadeia,token="id",cat="nome\_prog")

program meu\_prog ...

Cadeia	Token	Categoria	Tipo	Valor	...
meu_prog	id	nome_prog	-	-	...

59

## Tabela de símbolos

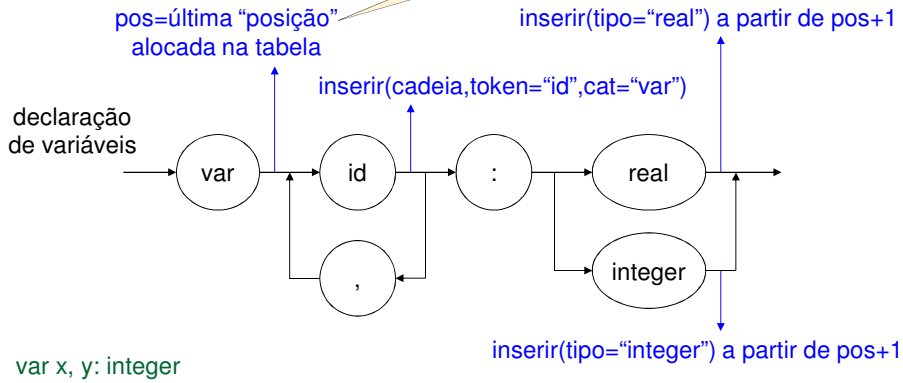


var x, y: integer

Cadeia	Token	Categoria	Tipo	Valor	...
meu_prog	id	nome_prog	-	-	...
x	id	var	integer		...
y	id	var	integer		...

# Tabela de símbolos

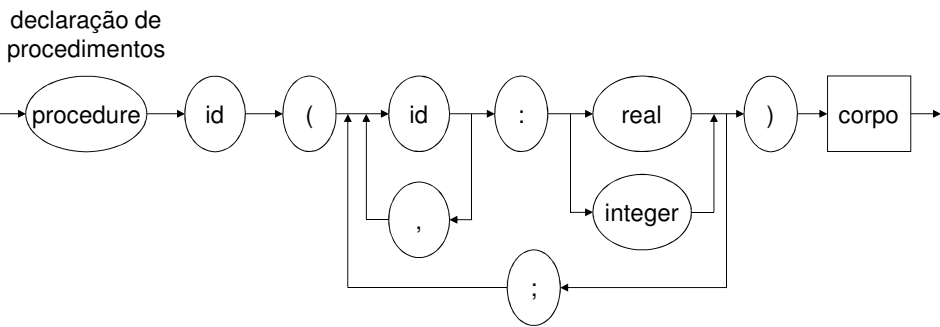
Dependente da forma de acesso à tabela



Cadeia	Token	Categoria	Tipo	Valor	...
meu_prog	id	nome_prog	-	-	...
x	id	var	integer		...
y	id	var	integer		...

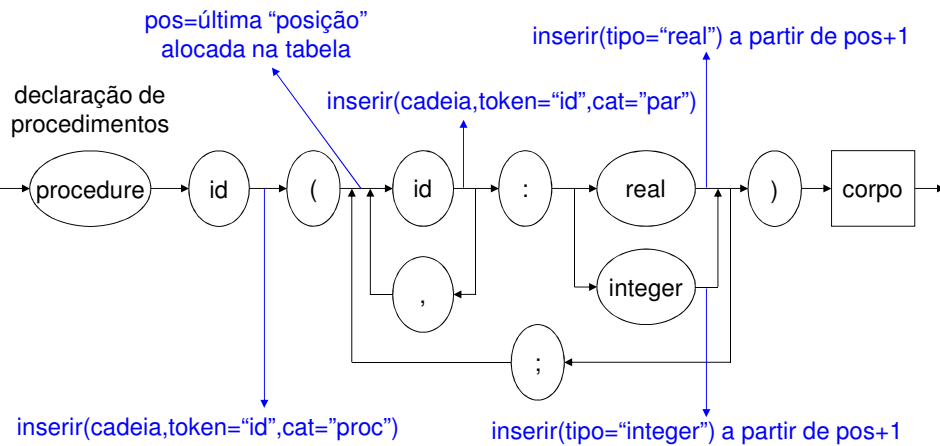
# Tabela de símbolos

- **Exercício:** inclua as funções adequadas



## Tabela de símbolos

- Exercício: inclua as funções adequadas



63

## Tabela de símbolos

- Exercício: inclua as funções adequadas

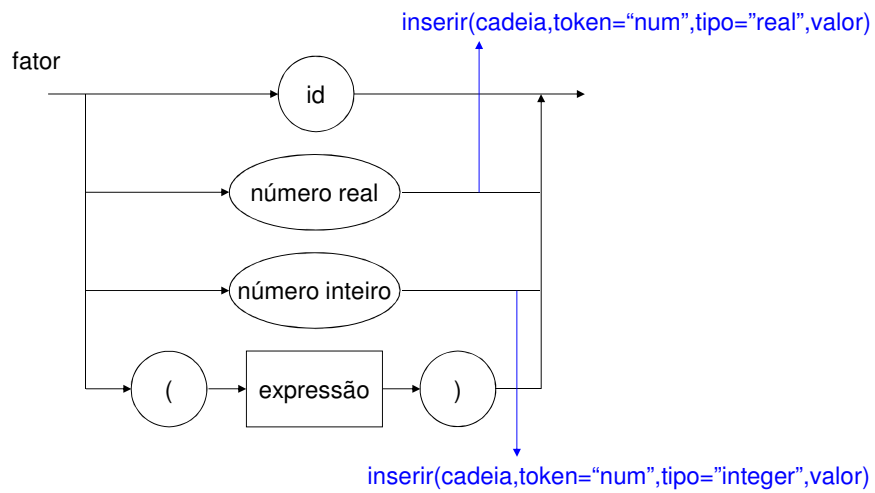
procedure meu\_proc(a: integer; b,c: real) ...

Cadeia	Token	Categoria	Tipo	Valor	...
meu_prog	id	nome_prog	-	-	...
x	id	var	integer		...
y	id	var	integer		...
meu_proc	id	proc	-	-	...
a	id	par	integer		...
b	id	par	real		...
c	id	par	real		...

64



## Tabela de símbolos



65

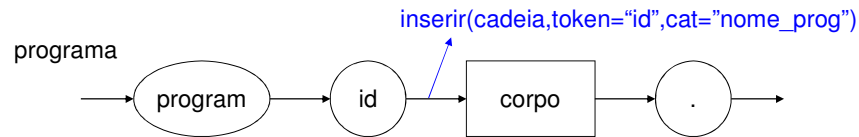
## Tabela de símbolos

`i:=1`

Cadeia	Token	Categoria	Tipo	Valor	...
meu_prog	id	nome_prog	-	-	...
x	id	var	integer		...
y	id	var	integer		...
meu_proc	id	proc	-	-	...
a	id	par	integer		...
b	id	par	real		...
c	id	par	real		...
1	num	-	integer	1	...

66

## Exemplo de procedimento



procedimento programa(Seg)

Início

```
se (simbolo=program) então obtem_simbolo(cadeia,simbolo)
senão ERRO(Seg+{id});
se (simbolo=id) então
    inserir(cadeia,"id","nome_prog")
    obtem_simbolo(cadeia,simbolo)
senão ERRO(Seg+P(corpo));
corpo(Seg+{.});
se (simbolo=simb_ponto) então obtem_simbolo(cadeia,simbolo)
senão ERRO(Seg);
```

fim

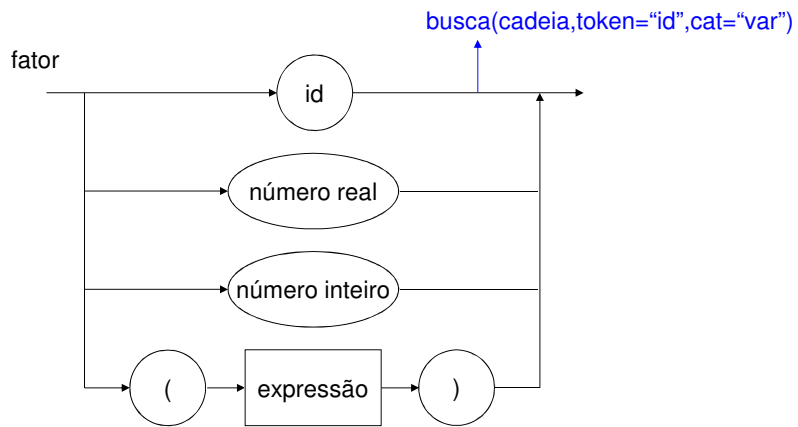
67

## Tabela de símbolos

- **Busca** de informação
  - Sempre que um elemento do programa é utilizado
    - fator e comando
  - Verifica-se se foi declarado, seu tipo, etc.

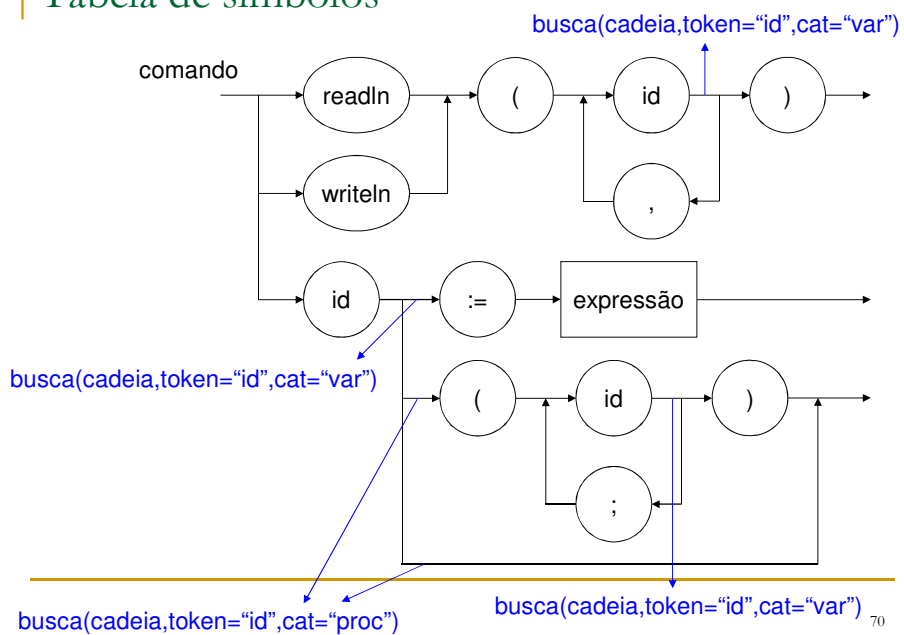
68

## Tabela de símbolos



69

## Tabela de símbolos



70

## Tabela de símbolos

- **Remoção** de elementos da tabela
  - Variáveis locais dos procedimentos
    - Atenção: parâmetros precisam ser mantidos

