unesp

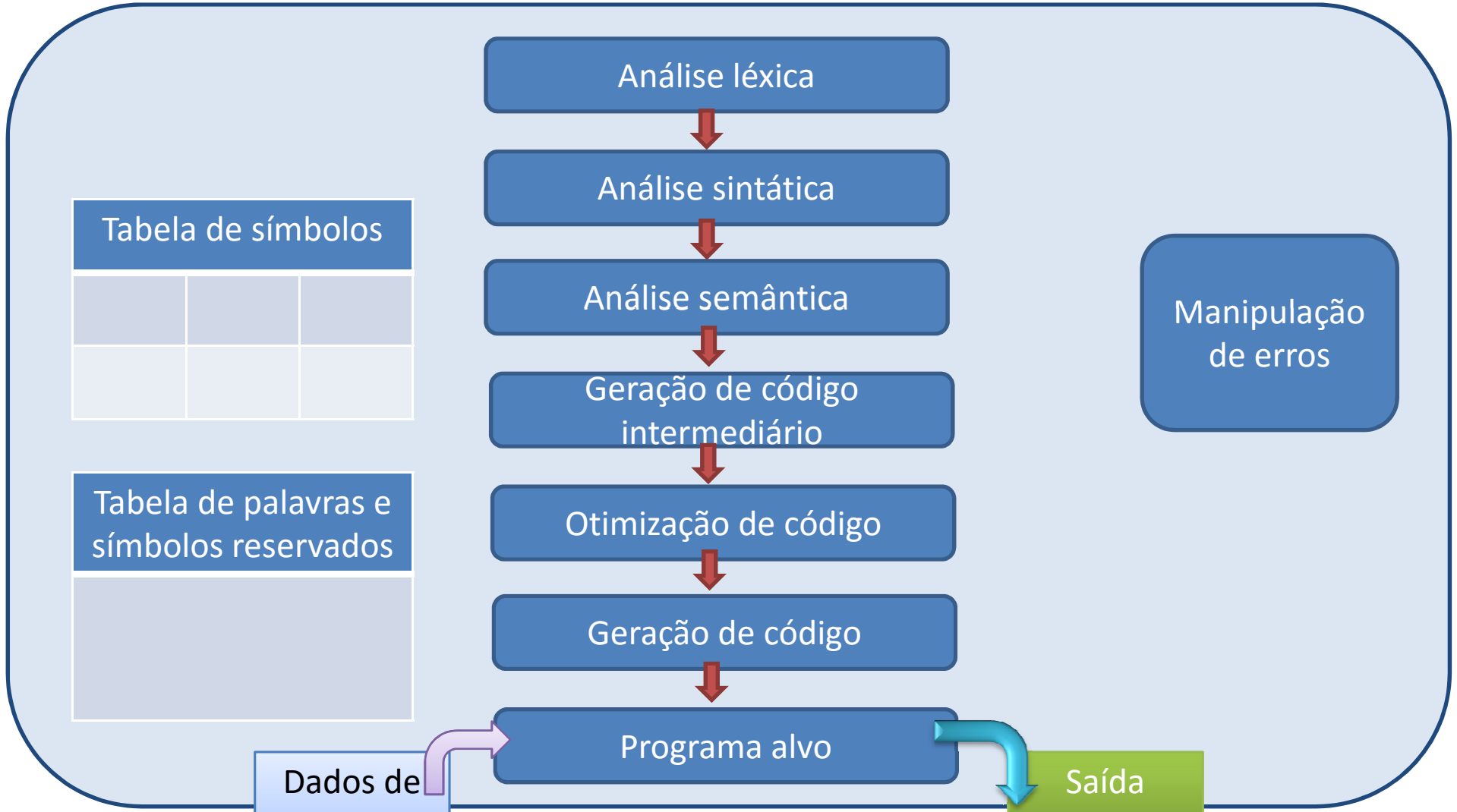
Compiladores

Aula 12

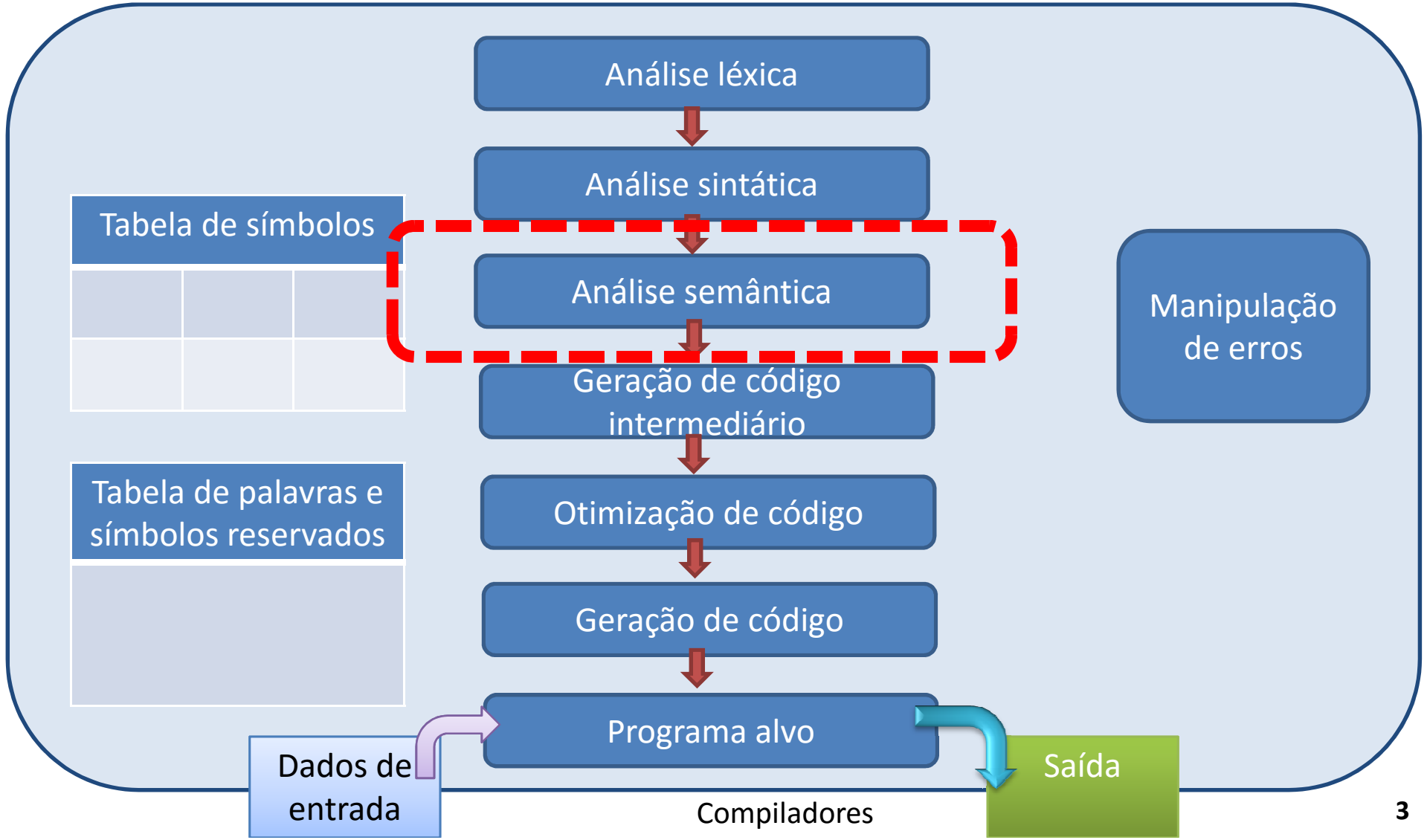
Celso Olivete Júnior

olivete@fct.unesp.br

Na aula de hoje



Na aula de hoje



Análise semântica

- Função: verificação do uso adequado
  - Análise contextual: declarações prévias de variáveis, procedimentos, etc.
  - Checagem de tipos: o tipo da variável é o correto para o operador?
  - Unicidade: o identificador (variável, rótulo) é único no escopo?
  - Coisas que vão além do domínio da sintaxe

Análise semântica

- Tipos de verificações semânticas
  - X foi declarado apenas uma vez?
  - X foi declarado/definido antes do seu primeiro uso?
  - X é declarado mas nunca foi utilizado?
  - A que declaração X se refere?
  - Os tipos de uma expressão são compatíveis?
  - As dimensões casam com o declarado?

Análise semântica

- Tipos de análise semântica
  - Estática, em tempo de compilação: linguagens tipadas, que exigem declarações
    - ✓ C, Pascal, etc.
  - Dinâmica, em tempo de execução: linguagens em que as variáveis são determinadas pelo contexto de uso
    - ✓ LISP, PROLOG, PHP

Análise semântica

- Devido às variações de especificação semântica das linguagens de programação, a análise semântica
  - Não é tão bem formalizada
  - Não existe um método ou modelo padrão de representação do conhecimento
  - Não existe um mapeamento claro da representação para o algoritmo correspondente
- Muitas vezes a análise é artesanal, dependente da linguagem de programação

Análise semântica
formalização e implementação

1. Semântica (tradução) dirigida pela sintaxe
   - Conteúdo semântico fortemente relacionado à sintaxe do programa
   - Maioria das linguagens de programação modernas utilizam
2. Tabela de símbolos

- Em geral, a semântica de uma linguagem de programação não é especificada
  - O projetista do compilador tem que analisar e extrair a semântica

Análise semântica
formalização e implementação

1. Semântica dirigida pela sintaxe feita com o uso de gramáticas de atributos
   - diz que ações serão realizadas considerando
     - ✓ Comportamento semântico das operações
     - ✓ Checagem de tipos
     - ✓ Manipulação de erros
     - ✓ Tradução do programa

Análise semântica
formalização e implementação

2. Tabela de símbolos

- Estrutura auxiliar essencial para a análise semântica
- Permite saber durante a compilação de um programa o tipo e endereço de seus elementos, escopo destes, número e tipo dos parâmetros de um procedimento, etc.
- Cada categoria de *token* tem atributos/informações diferentes associadas

Lexema	*Token*	Categoria	Tipo	Valor	Escopo	Utilizada
i	id	var	integer	1	...	S
fat	id	proc	-	-	...	...
2	num	-	integer	2	...	N
...						

Formalização e implementação

- Assim como a sintaxe, a semântica precisa ser formalizada/descrita antes de ser implementada
  - Sintaxe: por exemplo, BNF com procedimentos recursivos
- Formas de descrever a semântica:
  - Pode ser especificada informalmente (e geralmente é artesanal)
    - descrições em manuais de cada linguagem
  - ou formalmente
    - Gramática de atributos

Gramática de atributos

Gramática de atributos

- É a tradução de uma linguagem guiada pela GLC que a descreve
  - ✓ O processo ocorre junto com a análise sintática
  - ✓ São agregados **atributos** aos símbolos **não- terminais** da gramática
  - ✓ São atreladas **ações semânticas** às **produções** da gramática
  - ✓ As **ações semânticas** associadas às produções **computam os valores** dos **atributos dos símbolos**

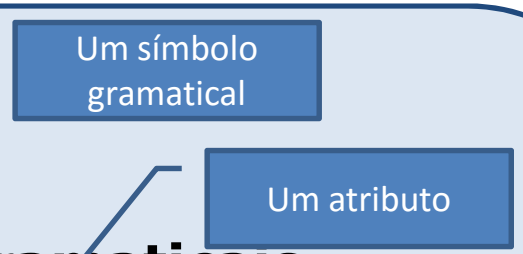
Gramática de atributos

❑Gramática de atributos

- ➢ Atributos **associados aos símbolos gramaticais**
  - ✓ Por exemplo, valor e escopo
    - ❖ **Representados na forma:** x.valor, x.escopo

Gramática de atributos

❑Gramática de atributos



- Atributos **associados aos símbolos gramaticais**
  - ✓ Por exemplo, valor e escopo
    - ❖ **Representados na forma:** 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

Gramática de atributos

- **Atributos:**
  - Cada símbolo pode possuir um ou mais atributos associados
  - Os atributos podem ser de qualquer tipo
    - Uma cadeia, um número, um tipo, uma localização de memória,..
  - O valor de um atributo é definido por uma regra semântica associada a produção que contém o símbolo dono do atributo
  - Pode ocorrer de uma regra de produção não apresentar nenhuma regra semântica

Gramática de atributos

- Exemplo de gramática de atributos

exp → exp + termo | exp - termo | termo
termo → termo * fator | termo div fator | fator
fator → (exp) | num

- 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**

Regras gramaticais	Regras semânticas
exp→exp$_1$+termo	exp.val=exp$_1$.val+termo.val
exp→exp$_2$-termo	exp.val=exp$_2$.val-termo.val
exp→termo	exp.val=termo.val
termo→termo$_1$*fator	termo.val=termo$_1$.val*fator.val
termo→termo$_2$ div fator	termo.val=termo$_2$.val/fator.val
termo→fator	termo.val=fator.val
fator→ (exp)	fator.val=(exp.val)
fator→num	fator.val=num.val

Gramática de atributos

- Atenção
  - Nem todo símbolo gramatical tem atributos
  - Pode haver manipulação de mais de um atributo em uma mesma regra e para um mesmo símbolo
  - Pode não haver regras semânticas para uma regra sintática

Cálculo dos atributos

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

Cálculo dos 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
- Em compilação, a tabela de símbolos é utilizada, junto com retorno de parâmetros/variáveis para checagem de tipos e sensitividade ao contexto.

Questões de implementação

- A implementação da análise semântica não será realizada com o uso de gramática de atributos

- Serão implementadas regras semânticas *Ad hoc*
  - Análise semântica "comandada" pela análise sintática
    - checagem contextual atrelada ao procedimento sintático e tabela de símbolos
    - checagem de tipo
      - Objetivo: verificar se o tipo de uma expressão "casa" com o esperado em seu contexto.
      - Exemplo: Operador ***div*** requer tipos inteiros, portanto, a checagem de tipos deve verificar se os operandos são inteiros

Questões de implementação

- *Ad hoc*
  - Análise semântica "comandada" pela análise sintática
  - Compilador de uma única passagem

Tabela de símbolos

❑ 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

Tabela de símbolos

- Estrutura principal da compilação
- Captura a sensitividade 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

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

Atributo para verificar se a variável foi declarada mas não foi utilizada

Cadeia	*Token*	Categoria	Tipo	Valor	Escopo	Utilizada
i	id	var	integer	1		S
fat	id	proc	-	-		...
2	num	-	integer	2		N
...						

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.), se foi utilizada, etc.
- Para vetor, ainda seriam necessários atributos de tamanho do vetor, o valor de seus limites, etc.

Tabela de símbolos

- Principais operações efetuadas na tabela de símbolos
  - Inserir: armazena na tabela informações fornecidas pelas declarações no programa
  - Busca: recupera da tabela as 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 operações citadas são inseridas diretamente nas chamadas dos procedimentos

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 e escopo)
  - Remover um elemento quando este não se faz mais necessário ao programa – variáveis locais ao procedimento

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

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)
      - Várias tabelas: Diferentes declarações têm diferentes informações/atributos (por exemplo, variáveis não têm número de argumentos, enquanto procedimentos têm)

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

Tabela de símbolos

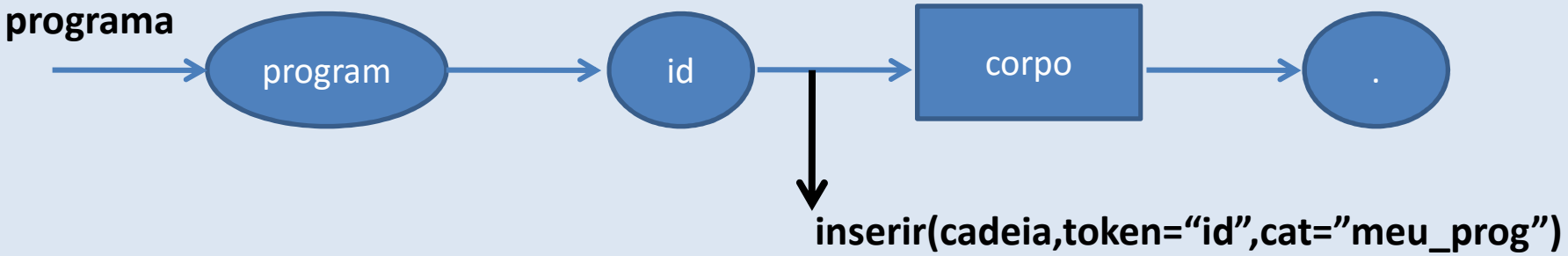
- Possibilidades para tratamento de escopos
  - Inclusão de um campo a mais na tabela de símbolos indicando o nível da variável no programa
    - ✓ Controle do nível durante a compilação do programa
      - ❖ Quando se chama um procedimento (ou função), faz-se nível=nível+1
      - ❖ Quando se sai de 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

Tabela de símbolos

- Sub-rotinas de **inserção**, **busca** e **remoção** inseridas diretamente na análise sintática

Tabela de símbolos / Grafos sintáticos

- Inserção de elementos na tabela
  - Declaração, principalmente

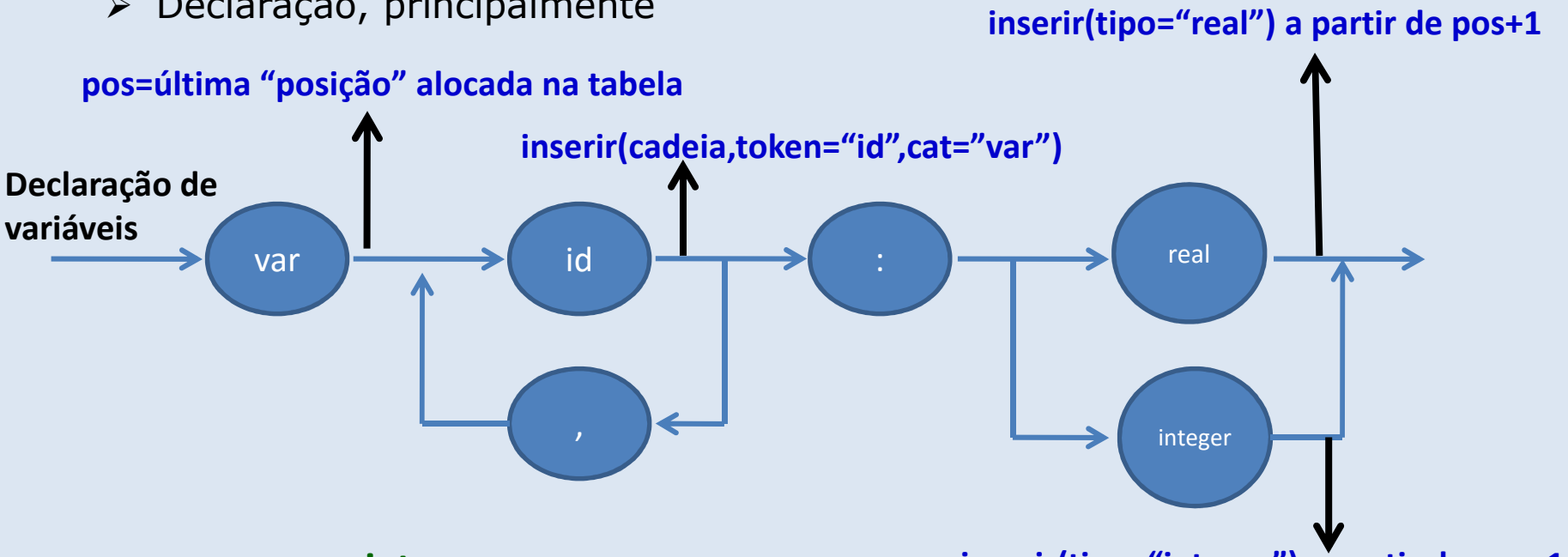


program meu_progr...

Cadeia	*Token*	Categoria	Tipo	Valor	Escopo	Utilizada
meu_prog	id	meu_prog	-	-	...	

Tabela de símbolos

- Inserção de elementos na tabela
  - Declaração, principalmente



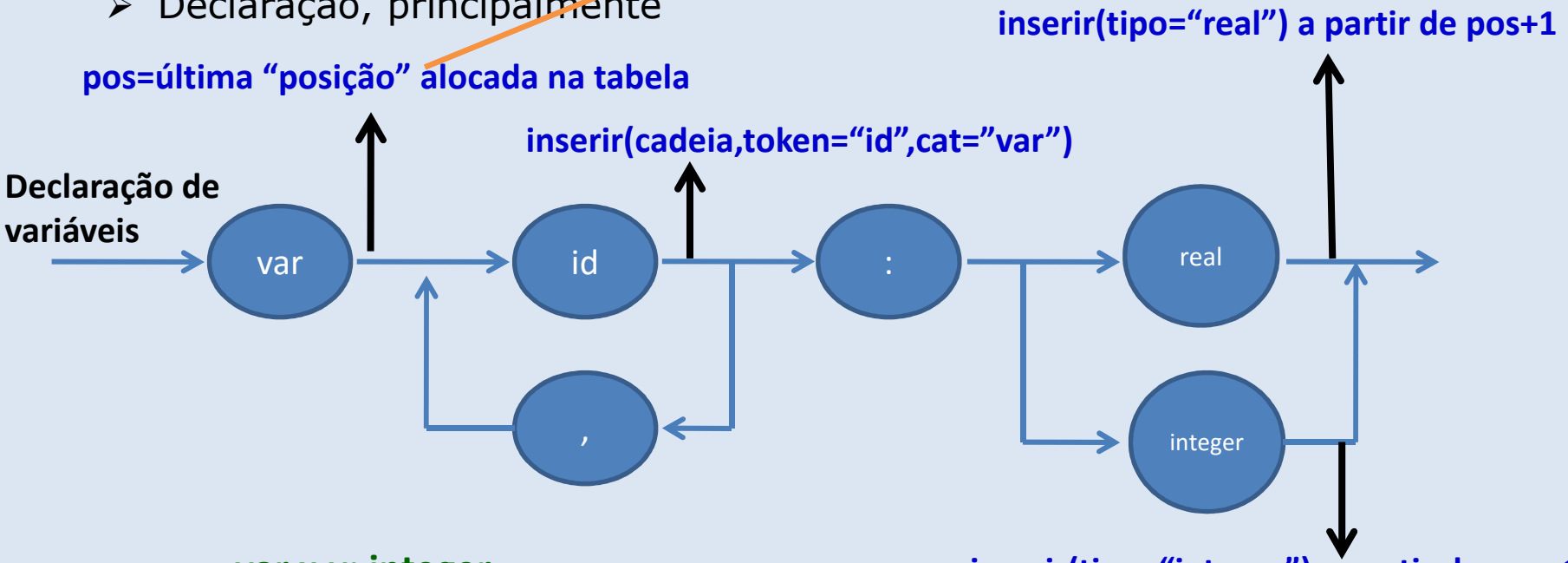
var x,y: integer

Cadeia	*Token*	Categoria	Tipo	Valor	Escopo	Utilizada
meu_prog	id	meu_prog	-	-	...	
x	id	var	integer			
y	id	var	integer			

Tabela de símbolos

Dependente da forma de acesso à tabela

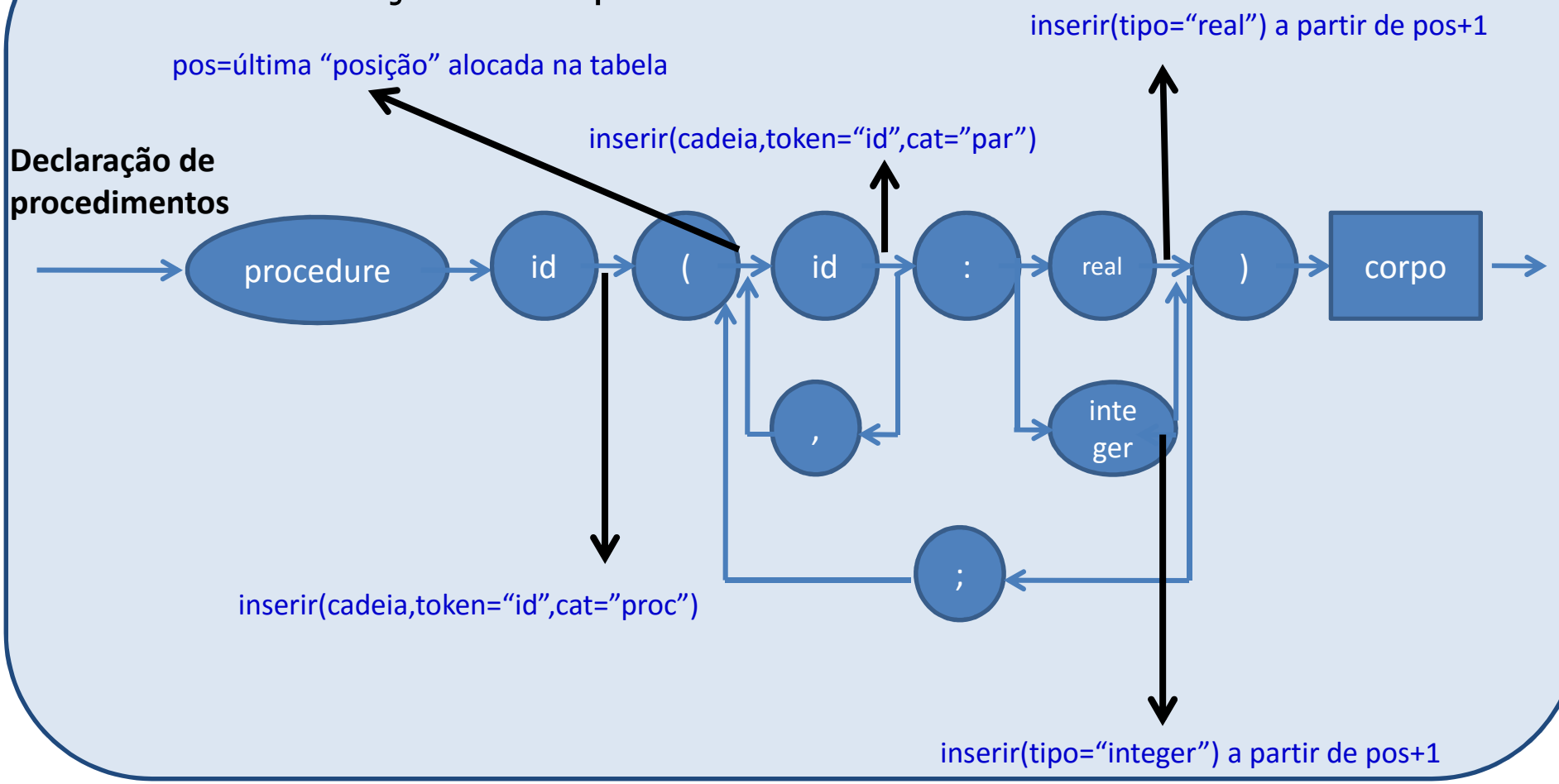
- Inserção de elementos na tabela
  - Declaração, principalmente



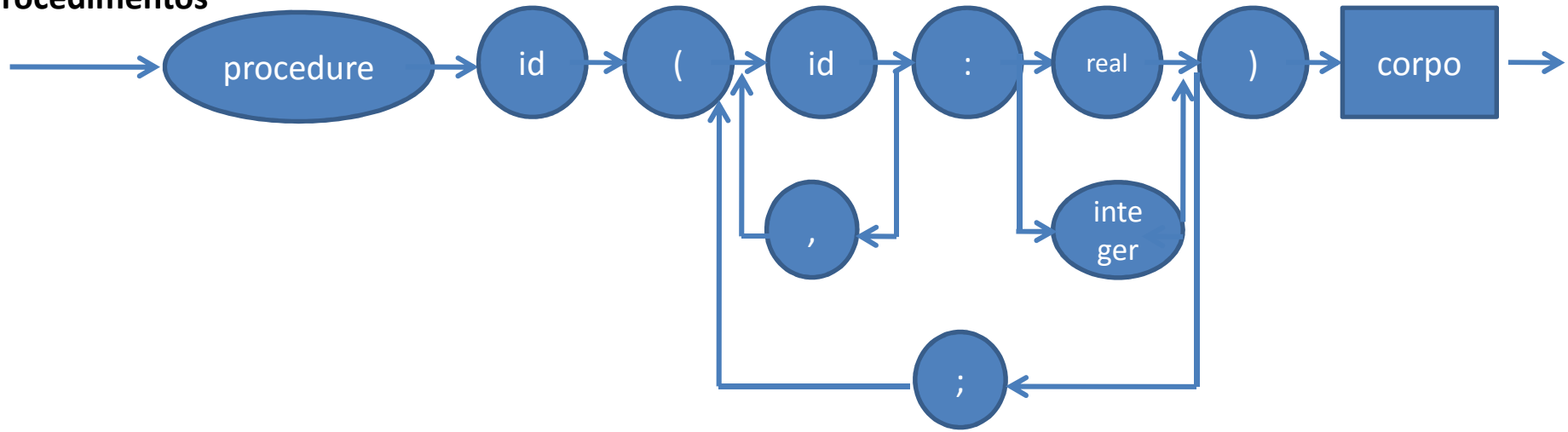
Cadeia	*Token*	Categoria	Tipo	Valor	Escopo	Utilizada
meu_prog	id	meu_prog	-	-	...	
x	id	var	integer		...	
y	id	var	integer		...	

Tabela de símbolos
Exercício II

- Inclua as funções adequadas



Declaração de procedimentos



procedure meu_proc(a: integer; b,c: real) ...

Cadeia	*Token*	Categoria	Tipo	Valor	Escopo	Utilizada
meu_prog	id	meu_prog	-	-	...	
x	id	var	integer		...	
y	id	var	integer		...	
meu_proc	id	proc	-	-	procedimento	
a	id	par	integer		parâmetro	
b	id	par	real		parâmetro	
c	id	par	real		parâmetro	

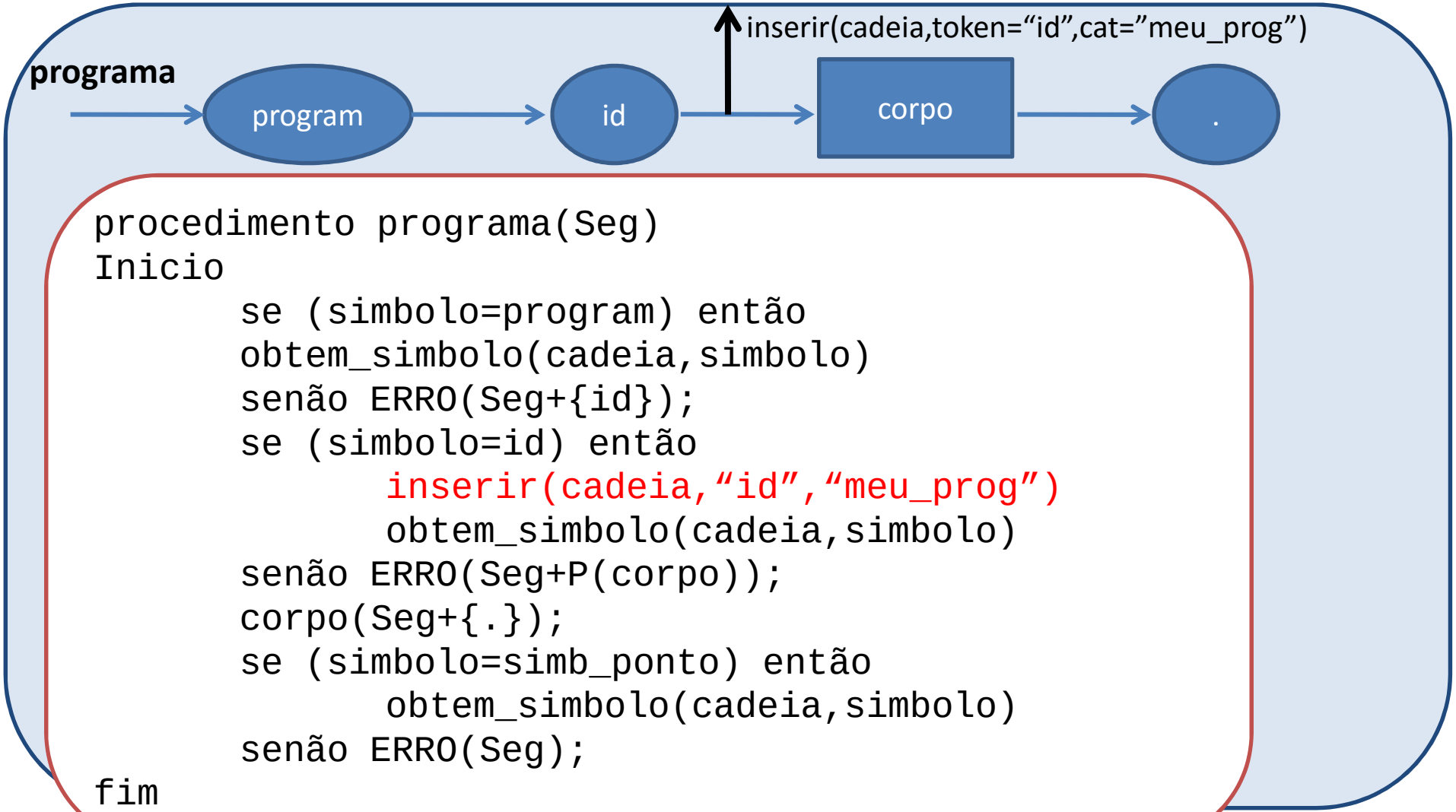
Tabela de símbolos

- Inclua as funções adequadas

i:=1

Cadeia	*Token*	Categoria	Tipo	Valor	...
meu_prog	id	meu_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		...
i	num	-	integer	1	...

Exemplo de procedimento




```
procedimento programa(Seg)
Inicio
        se (simbolo=program) então
        obtem_simbolo(cadeia,simbolo)
        senão ERRO(Seg+{id});
        se (simbolo=id) então
                inserir(cadeia,"id","meu_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
```

Tabela de símbolos

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

Tabela de símbolos

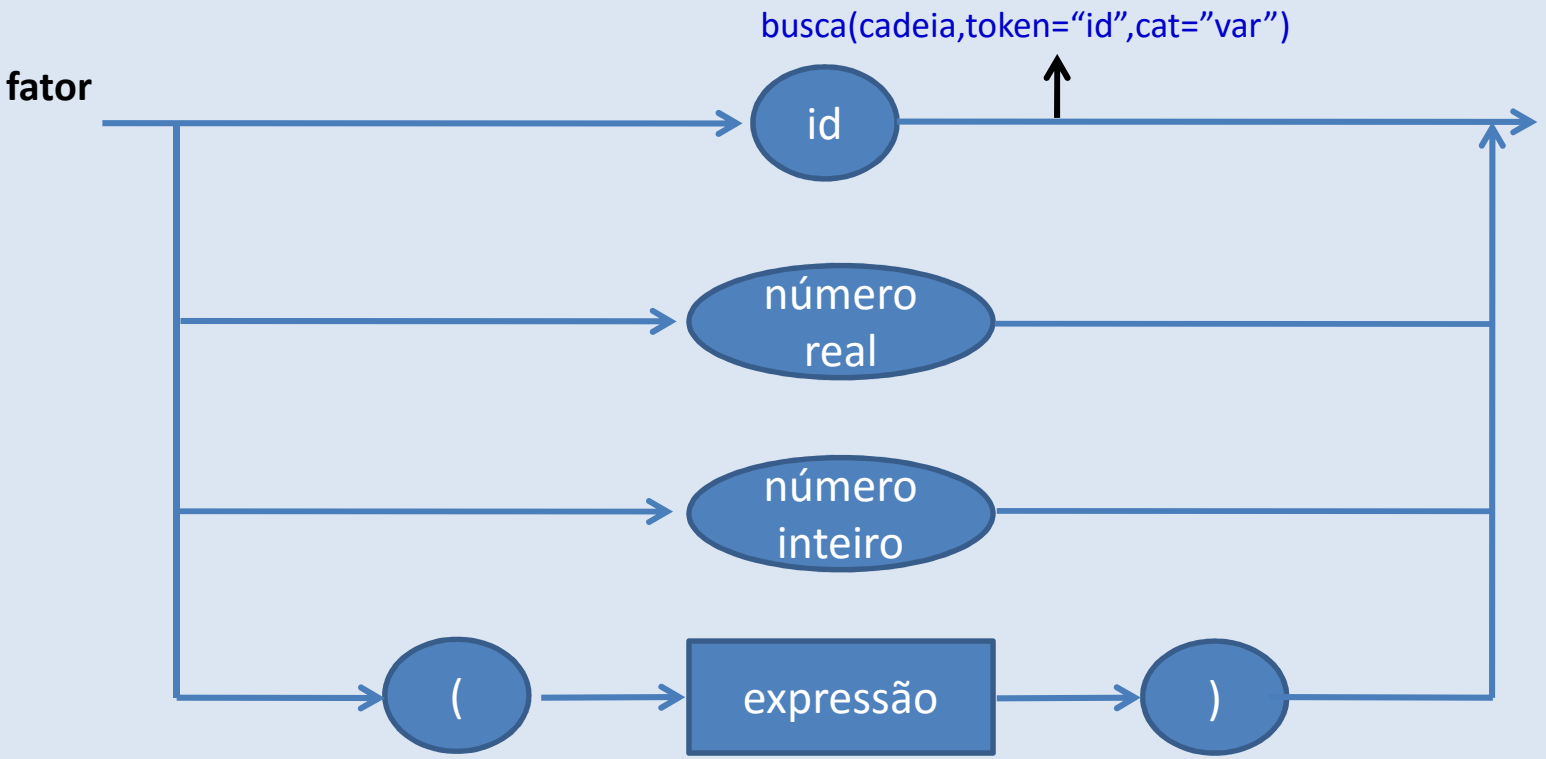
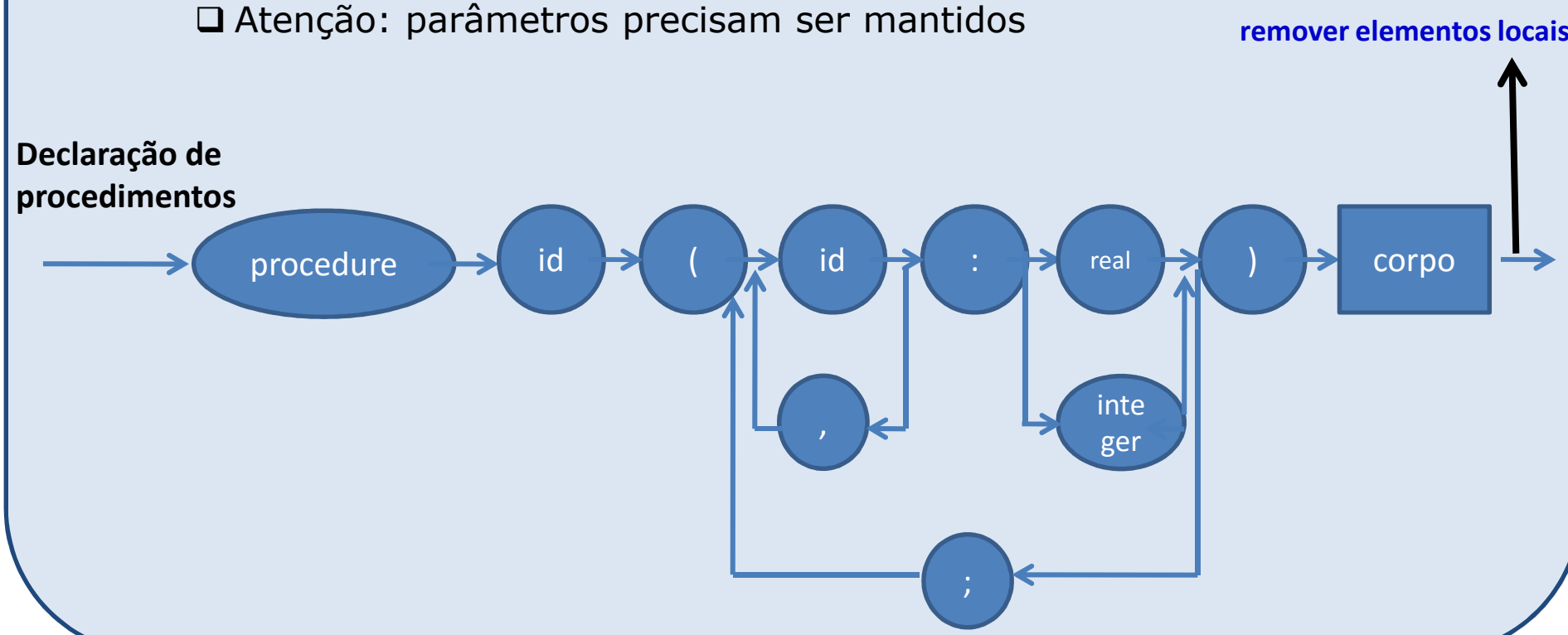


Tabela de símbolos

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



Principais erros semânticos na LALG

- Variável ou procedimento não declarado
- Variável ou procedimento declarado mais de uma vez
- Incompatibilidade de parâmetros formais e reais: número, ordem e tipo
- Uso de variáveis de escopo inadequado
- Atribuição de um real a um inteiro
- Divisão que não é entre números inteiros
- Variável declarada e nunca utilizada
- Read e write com variáveis de tipo diferentes
- Tratamento de escopo
  - Erro: variável local a um procedimento utilizada no programa principal

```
procedimento dc_v(S)
begin
    se (simb=var) então obter_símbolo()
    senão
            imprimir("Erro: var esperado");
            ERRO(Primeiro(variaveis)+S); //consome até encontrar ID
    variaveis({:}+S);
    se (simb=simb_dp) então obter_símbolo()
    senão
            imprimir("Erro: ':' esperado");
            ERRO(Primeiro(tipo_var)+S); //consome até encontrar integer ou real
    tipo_var({;}+S);
    se (simb=simb_pv) então obter_símbolo()
    senão
            imprimir("Erro: ';' esperado");
            ERRO(Primeiro(dc_v)+S); //consome até encontrar ;
    dc_v(S);
end;
```

Rotina Declaração de Variáveis

```
<DC_V> ::= var <VARIAVEIS> : <TIPO_VAR> ; <DC_V> | λ
<TIPO_VAR> ::= integer | real
<VARIAVEIS> ::= <ID> <MAIS_VAR>
<MAIS_VAR> ::= , <VARIAVEIS> | λ
```

var a,b: integer

```
procedimento variaveis(S)
begin
se (simb=id)
    então
    se busca(cadeia, token="id", cat="var")==false
            então inserir(cadeia,token="id",cat="var")
            senão ERRO("identificador já declarado")
    obtem_simbolo(cadeia,simbolo)
    enquanto (simbolo=simb_virgula) faça
            obtem_simbolo(cadeia,simbolo)
            se (simb=id)
            então
                    se busca(cadeia, token="id", cat="var")==false
                            então inserir(cadeia,token="id",cat="var")
                            senão ERRO("identificador já declarado")
                    obtem_simbolo(cadeia,simbolo)
            fim-então
            senão ERRO(S+{simb_virgula,simb_dois_pontos});
    fim-enquanto
end;
```

Rotina Declaração de Variáveis

```
<DC_V> ::= var <VARIAVEIS> : <TIPO_VAR> ; <DC_V> | λ
<TIPO_VAR> ::= integer | real
<VARIAVEIS> ::= <ID> <MAIS_VAR>
<MAIS_VAR> ::= , <VARIAVEIS> | λ
```

var a,b: integer

Cadeia	*Token*	Categoria	Tipo	Valor	Utilizada	Escopo
a	id	var	...	0	N	0
b	id	var	...	0	N	0

Rotina Declaração de Variáveis

```
<DC_V> ::= var <VARIAVEIS> : <TIPO_VAR> ; <DC_V> | λ
<TIPO_VAR> ::= integer | real
<VARIAVEIS> ::= <ID> <MAIS_VAR>
<MAIS_VAR> ::= , <VARIAVEIS> | λ
```

```
procedimento tipo_var(S)
begin
se (simb=integer)
    então tab_simb_alterar_tipo(cadeia, token="id", cat="var", tipo="integer")
    senão tab_simb_alterar_tipo(cadeia, token="id", cat="var", tipo="real")
end;
```

Cadeia	*Token*	Categoria	Tipo	Valor	Utilizada	Escopo
a	id	var	**integer**	0	N	0
b	id	var	**integer**	0	N	0

```
procedimento cmd_IO(S)
início
se (simb=readln) ou (simb=writeln)
                então obtem_simbolo(cadeia,simbolo)
                senão ERRO(S+{simb_abre_par})
se (simb=simb_abre_par)
                então obtem_simbolo(cadeia,simbolo)
                senão ERRO(S+{id})
se (simb=id)
     então
                se busca(cadeia, token="id", cat="var")==false
                                então ERRO("identificador não declarado")
                                senão tipo1=recupera_tipo(cadeia, token="id", cat="var")
                obtem_simbolo(cadeia,simbolo)
                enquanto (simbolo=simb_virgula) faça
                                obtem_simbolo(cadeia,simbolo)
                                se (simb=id)
                                então
                                                se busca(cadeia, token="id", cat="var")==false
                                                então ERRO("identificador não declarado")
                                                senão tipo2=recupera_tipo(cadeia, token="id", cat="var");
                                                se (tipo1<>tipo2)
                                                                então ERRO("tipos incompatíveis");
                                                obtem_simbolo(cadeia,simbolo)
                                fim-então
                senão ERRO(S+{simb_virgula,simb_fecha_par});
                fim-enquanto
    fim-então
    senão ERRO(S+{simb_fecha_par})
se (simb=simb_fecha_par)
     então obtem_simbolo(cadeia,simbolo)
     senão ERRO(S)
fim
```

Rotina Entrada/Saída

```
<cmd-IO> ::=readln(<lista_var>)
          | writeln ( <lista_var> )
<lista_var> ::= ident <mais_var>
<mais_var> ::= , <lista_var> |λ
```

```
procedure termo (var t: string);
var t1, t2: string;
begin
Fator (t);
Enquanto simbolo in [*,div, and] faça
begin
    s1:= simbolo;
    simbolo:= analex(s);
    Fator(t1);
    Caso s1 seja
        * : t2 := 'inteiro';
        div: t2 := 'inteiro';
        and : t2 := 'booleano'
    end;
    Se (t <> t1) ou (t <> t2) então erro('incompatibilidade de tipos')
    end
end;
```

Rotina Termo

```
21. <termo> ::=  <fator>
        {(*| div| and)<fator>}
22.<fator> ::= <variavel>
               | <número>
               | (<expressão>)
               | not <fator>
```

```
procedure fator (var t: string);
Inicio
Caso simbolo seja
Número: {t:=inteiro;simbolo := analex(s);}
Identificador: {Busca(Tab: TS; id: string; ref: Pont_entrada; declarado: boolean);
        Se declarado = false então erro;
        Obtem_atributos(ref: Pont_entrada; AT: atributos);
        Caso AT.categoria seja
            Variavel: {t:= AT.tipo; simbolo := analex(s);}
            parametro: {t:= AT.tipo; simbolo := analex(s);}
        Else erro;
        fim caso;
Cod_abre_par: {simbolo := analex(s); expressao(t); se simbolo <>
                    Cod_fecha_par  then erro; simbolo := analex(s);}
Cod_neg: {simbolo := analex(s); fator(t);  se t <> 'booleano' então erro
Else erro;
Fim caso;
```

Rotina Fator

```
22.<fator> ::=
  <variavel>
| <número>
| (<expressão>)
| not <fator>
```

Implementação

- Façam as rotinas expressão simples e expressão de forma equivalente a termo
- Desta forma expressão retornará um parâmetro (T) que é o tipo da expressão.
- Este parâmetro deve ser checado nos comandos if e while: t deve ser booleano;
- em atribuição o lado esquerdo deve ser compatível (estruturalmente) com o direito;
- os parâmetros dos procedimentos também devem ser checados, pois o tipo do parâmetro real deve ser compatível com o tipo do parâmetro formal.

Implementação

- O enunciado da Parte 3 do Projeto já está disponível na página da disciplina