Compiladores
Aula 7
Análise Léxica
A Linguagem *TINY*

Prof. Dr. Luiz Eduardo G. Martins

UNIFESP

A Linguagem *TINY*

- A linguagem *TINY* será usada como exemplo para ilustrar características essenciais de um compilador
  - Ver seção 1.7 do livro "Compiladores: Princípios e Práticas", Kenneth C. Louden

A Linguagem *TINY*

- Características gerais da linguagem TINY:
  - Sequências de declarações separadas por ponto-e-vírgula (com exceção da declaração final)
  - Variáveis são do tipo inteiro ou booleano
  - Variáveis declaradas pela atribuição de valores
  - Há duas declarações de controle: *if* e *repeat*
  - *If* tem como opcional uma parte *else*, e precisa terminar com a palavra-chave *end*

A Linguagem *TINY*

- Características gerais da linguagem TINY:
  - Há declarações para entrada e saída

    read

    write
  - Permite comentários - aparecer entre chaves
    - Não podem ser aninhados
  - Permite expressões aritméticas com inteiros

    + - * / (divisão inteira)
  - Uma expressão aritmética pode ter constantes, variáveis e parênteses

A Linguagem *TINY*

- Características gerais da linguagem TINY:
  - Permite expressões booleanas
    - Expressão booleana composta por uma comparação de duas expressões aritméticas
    - Operadores relacionais: `<` e `=`
  - A linguagem não contempla:
    - Funções e Procedimentos
    - Matrizes
    - Estruturas
    - Valores em ponto flutuante

A Linguagem *TINY*

- Exemplo de um programa em *TINY*

```
{ Sample program
  in TINY language -
  computes factorial
}
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact  { output factorial of x }
end
```

A Linguagem *TINY*

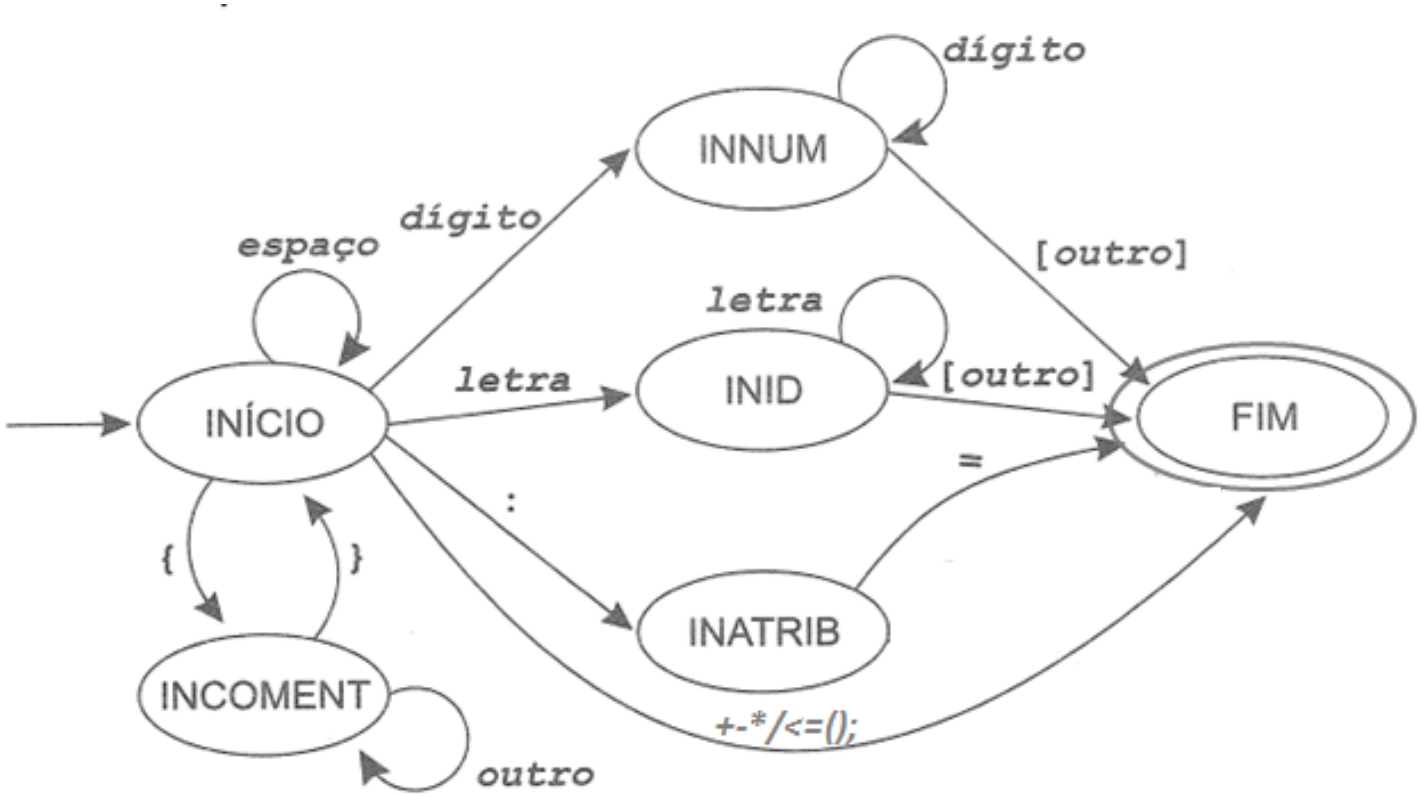
- Estrutura léxica da linguagem *TINY*

Tabela 2.1 Marcas da linguagem TINY

Palavras Reservadas	Símbolos Especiais	Outras
`if`	`+`	***número***
`then`	`-`	(1 ou mais
`else`	`*`	dígitos)
`end`	`/`	
`repeat`	`=`	
`until`	`<`	***identificador***
`read`	`(`	(1 ou mais
`write`	`)`	letras)
	`;`	
	`:=`	

A Linguagem *TINY*

- AFD para o analisador léxico de *TINY*



[outro] : não avança caractere

A Linguagem *TINY*

programa → *decl-seqüência*
decl-seqüência → *decl-seqüência* **;** *declaração* | *declaração*
declaração → *cond-decl* | *repet-decl* | *atrib-decl* | *leit-decl* | *escr-decl*
cond-decl → **if** *exp* **then** *decl-seqüência* **end**
| **if** *exp* **then** *decl-seqüência* **else** *decl-seqüência* **end**
repet-decl → **repeat** *decl-seqüência* **until** *exp*
atrib-decl → **identificador** **:=** *exp*
leit-decl → **read identificador**
escr-decl → **write** *exp*
exp → *exp-simples comp-op exp-simples* | *exp-simples*
comp-op → **<** | **=**
exp-simples → *exp-simples soma termo* | *termo*
soma → **+** | **-**
termo → *termo mult fator* | *fator*
mult → * | /
fator → **(***exp***)** | **número** | **identificador**

Figura 3.6 Gramática da linguagem TINY em BNF.

A Linguagem *TINY*

programa → *decl-seqüência*
decl-seqüência → *declaração* { **;** *declaração* }
declaração → *if-decl* | *repeat-decl* | *atribuição-decl* | *read-decl* | *write-decl*
if-decl → **if** *exp* **then** *decl-seqüência* [**else** *decl-seqüência*] **end**
repeat-decl → **repeat** *decl-seqüência* **until** *exp*
atribuição-decl → **identificador** **:=** *exp*
read-decl → **read** **identificador**
write-decl → **write** *exp*
exp → *simples-exp* [*comparação-op simples-exp*]
comparação-op → **<** | **=**
simples-exp → *termo* { *soma termo* }
soma → **+** | **-**
termo → *fator* { *mult fator* }
mult → ***** | **/**
fator → **(** *exp* **)** | **número** | **identificador**

Figura 4.9 Gramática da linguagem TINY em EBNF.

A Linguagem *TINY*

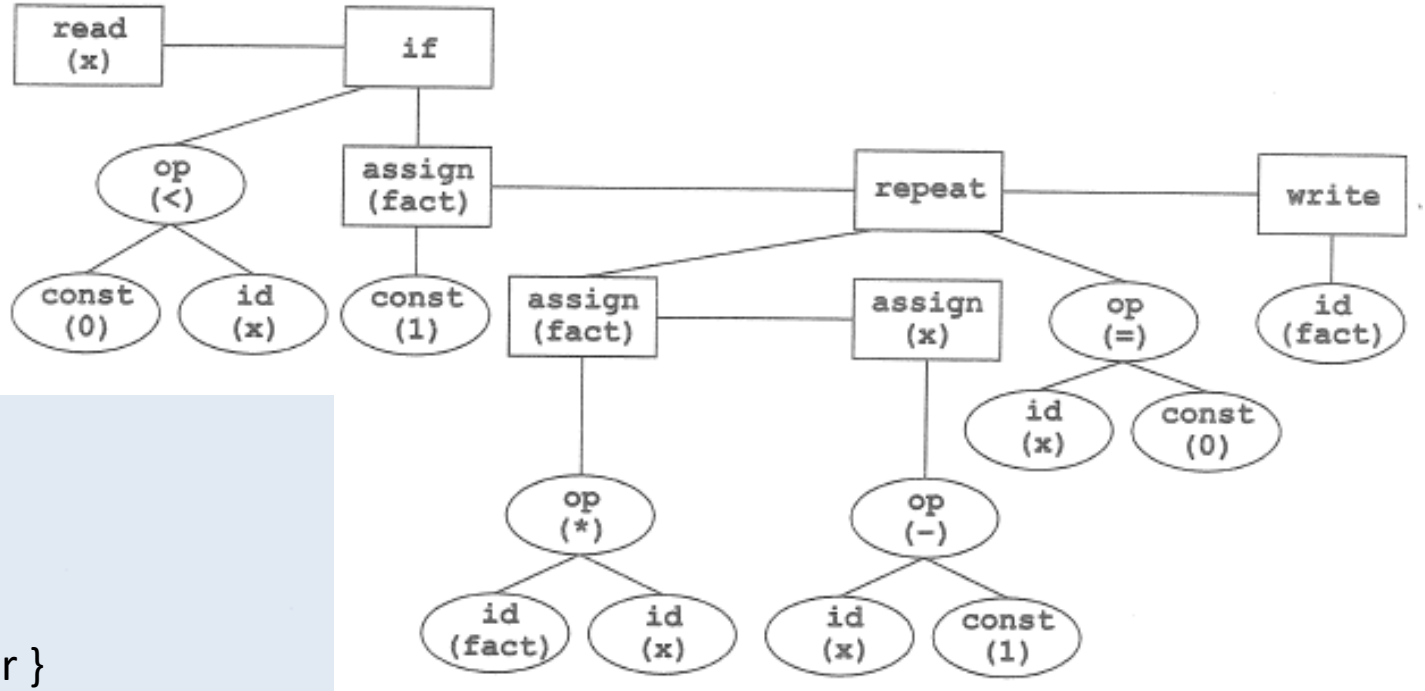


Figura 3.9 Árvore sintática para o programa TINY da Figura 3.8.

```
{ Sample program
  in TINY language -
  computes factorial
}
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact  { output factorial of x }
end
```

A Linguagem *TINY*

```
typedef enum {StmtK,ExpK} NodeKind;
typedef enum {IfK,RepeatK,AssignK,ReadK,WriteK}
             StmtKind;
typedef enum {OpK,ConstK,IdK} ExpKind;

/* ExpType é utilizado para verificação de tipos */
typedef enum {Void,Integer,Boolean} ExpType;

#define MAXCHILDREN 3

typedef struct treeNode
   { struct treeNode * child[MAXCHILDREN];
     struct treeNode * sibling;
     int lineno;
     NodeKind nodekind;
     union { StmtKind stmt; ExpKind exp;} kind;
     union { TokenType op;
             int val;
             char * name; } attr;
     ExpType type; /* para verificação de tipos de expressões */
   } TreeNode;
```

Figura 3.7 Declarações em C para um nó de árvore sintática TINY.

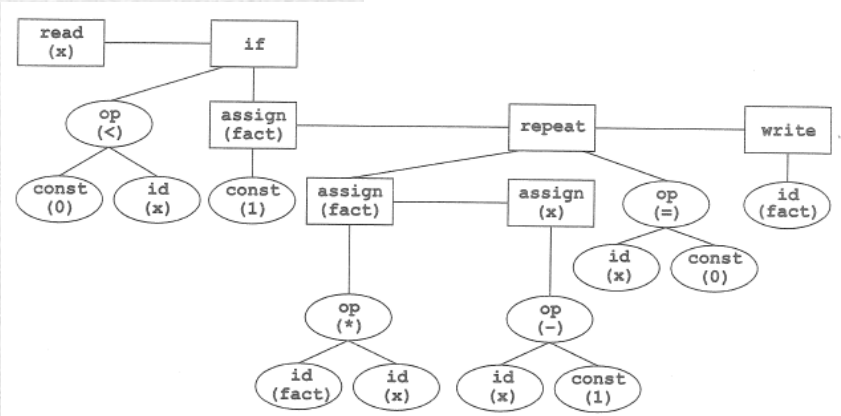


Figura 3.9 Árvore sintática para o programa TINY da Figura 3.8.

A Linguagem *TINY*

- Atividade

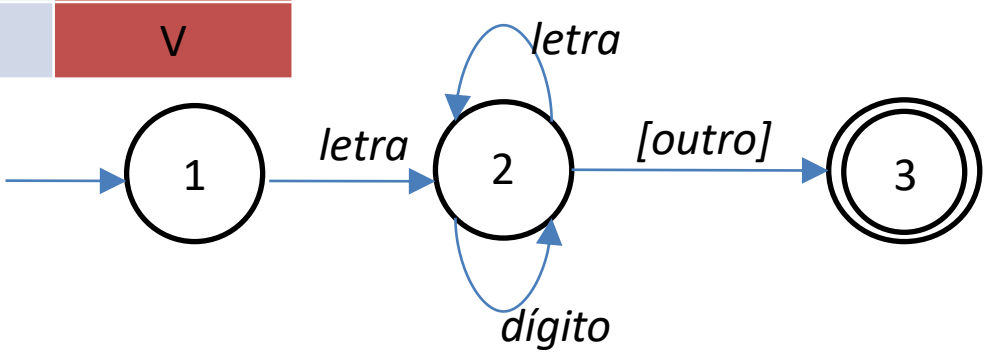
  a) Implementar um algoritmo (dirigido por tabela) que reconheça todos os *tokens* da linguagem *TINY*, o programa deve ler o arquivo sample.tny e fornecer a resposta idêntica ao arquivo sample.tokens

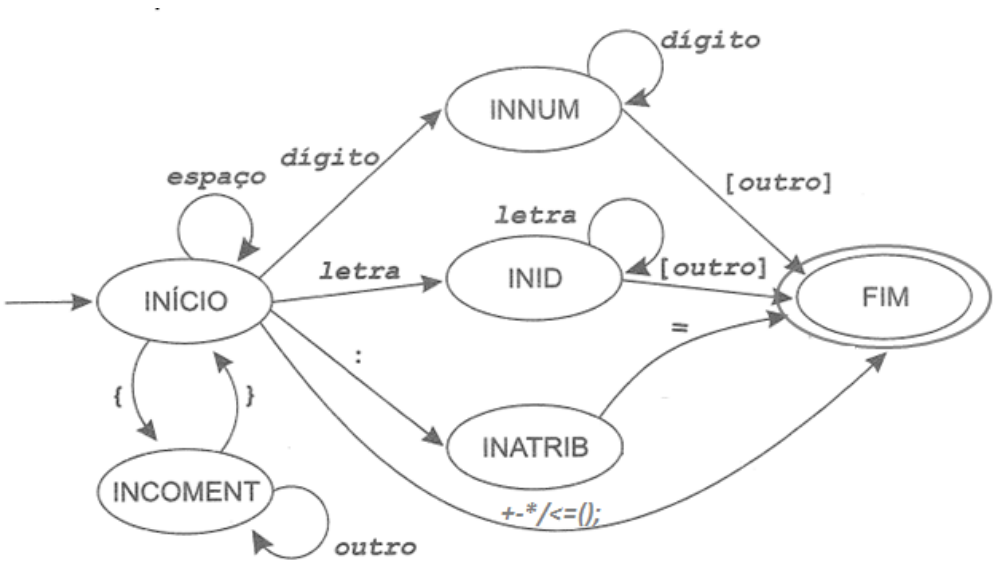
  - Inicialmente construa a tabela de transição de estados, de acordo com o exemplo visto em aula

A Linguagem *TINY*

- Exemplo:

	letra	dígito	outro	Aceitação
1	2 V			F
2	2 V	2 V	[3] F	F
3				V





A Linguagem *TINY*

- Bibliografia consultada

LOUDEN, K. C. **Compiladores: princípios e práticas.** São Paulo: Pioneira Thompson Learning, 2004