

# Curso: Engenharia de Computação

**Linguagens Formais e Compiladores**

Prof. Clayton J A Silva, MSc  
clayton.silva@professores.ibmec.edu.br



# Introduzindo a linguagem TINY – adaptada

- **Objetivo:** implementar as tarefas e estruturas de um compilador
- **Problema futuro:** escolha da linguagem de máquina como alvo do processo de compilação. Será que poderemos usar o set do microcontrolador Atmel 2560?
- Vamos introduzir a linguagem à medida que precisarmos – **nesta seção somente o suficiente para a varredura**

# Introduzindo a linguagem TINY - adaptada

- Alfabeto:

$$\Sigma = \{ '0', '1', \dots, '9', 'a-z', 'A-Z', '<', '>', '=', ';', '(', ')', '{', '}', '#' \},$$

, em que as aspas simples (') delimitam os caracteres do alfabeto

- Sequência de declarações separadas por ponto e vírgula
- Todas as variáveis são do tipo inteiro que não precisam ser declaradas, somente pela atribuição de valores
- Há apenas duas declarações de controle: ***if*** e ***while***; que contêm sequência de declarações
- Os *tokens* são de três tipos: **palavras reservadas**, **símbolos especiais**, **outros**

Palavras reservadas	Símbolos especiais	Outros
if else while read print main	+ - * / = < > ( ) ; { } #	Números – 1 ou mais dígitos Identificador – 1 ou mais letras

# Introduzindo a linguagem TINY – adaptada

- Expressões são limitadas a expressões **aritméticas** de inteiros e **booleanos**
- Expressões booleanas são compostas pela comparação de duas expressões aritméticas – **só** podem ser **usadas como testes** nas instruções compostas de controle de fluxo
- Expressões aritméticas podem conter **constantes** ou **variáveis**, parênteses e os operadores de inteiros

# Código na linguagem TINY - adaptada

```
main ( ) {  
  read x; #inteiro de entrada#  
  if x > 0 #nao calcula se x<=0#  
    fact = 1;  
    while x > 0 {  
      fact = fact * x  
      x = x - 1  
    }  
    print x  
}
```

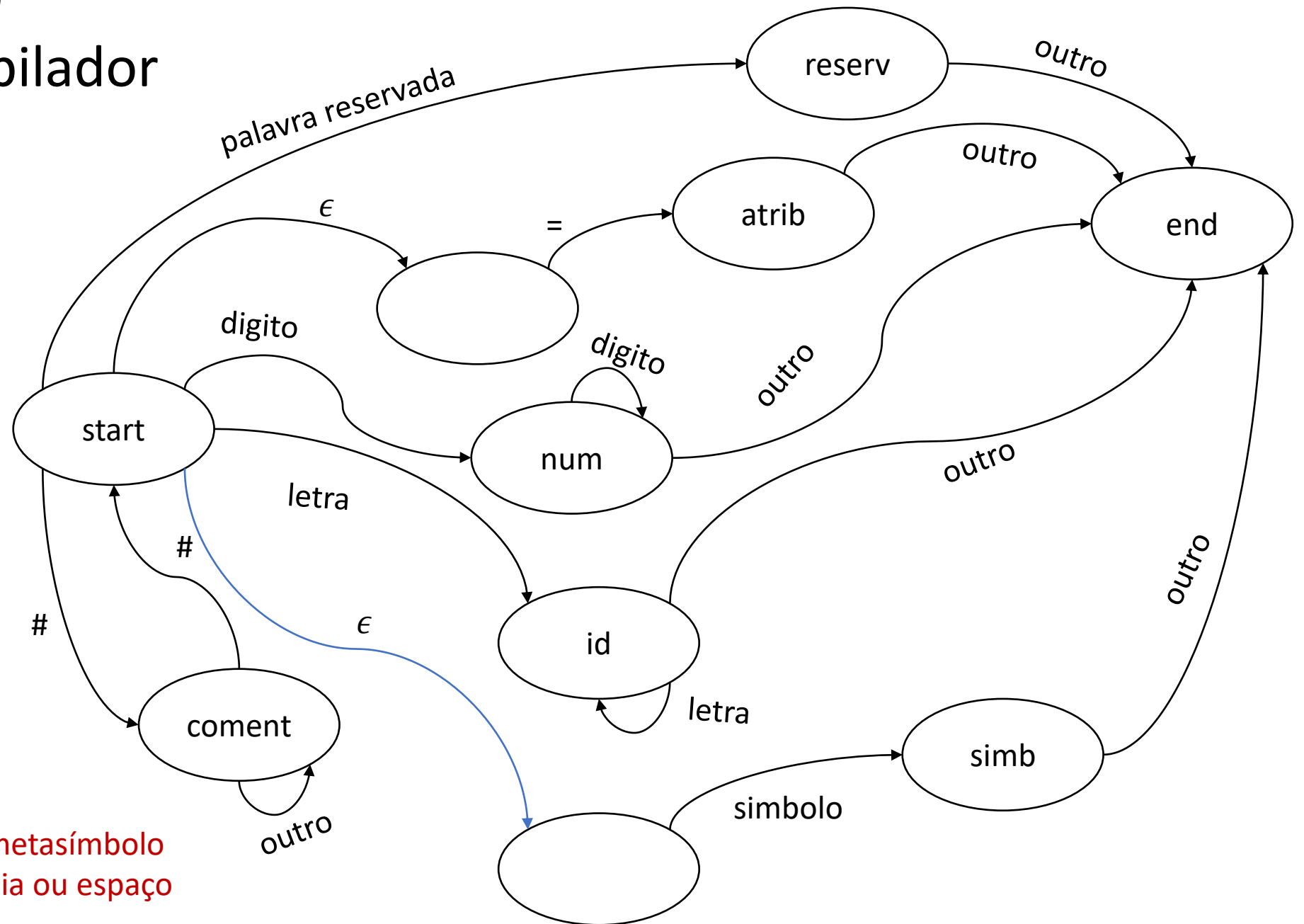
# Introduzindo a linguagem TINY - adaptada

- Espaços são compostos por espaços em branco, tabulações e mudanças de linha
- Além dos *tokens* definidos, a implementação pode usar *tokens* de controle, por exemplo ENDFILE , que marca um final de arquivo e ERRO, que marca um erro na varredura

# TINY (adaptada)

## Projeto do compilador

### NFA



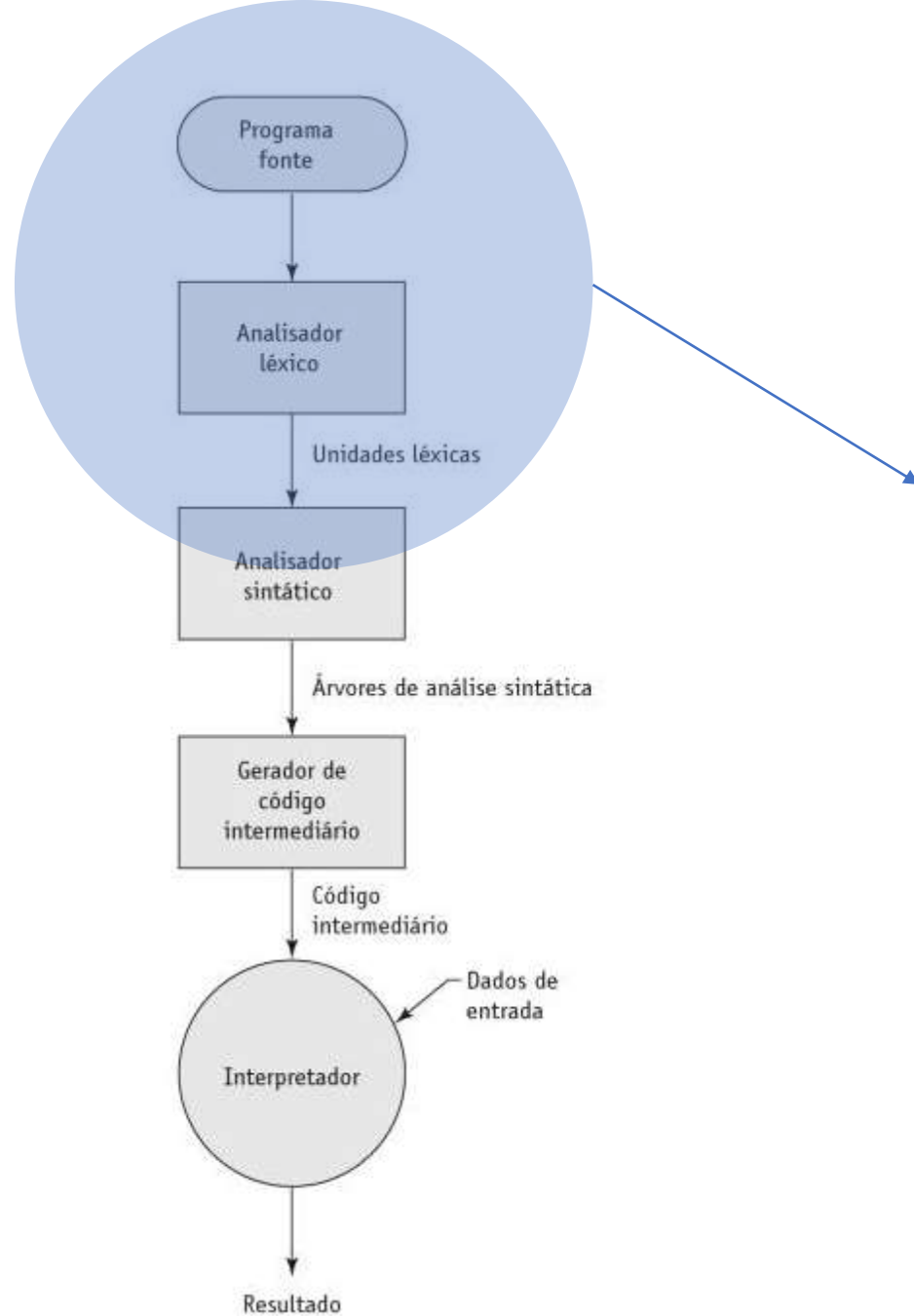
Obs. Lembrando que  $\epsilon$  é um metassímbolo do NFA, que define cadeia vazia ou espaço



# TINY (adaptada)

## Projeto do compilador

### Contexto



# registro de *token*

*typedef enum*

```
{RESERVADA, SIMBOLO, OUTRO}
```

```
TokenTipo;
```

*typedef struct*

```
{ TokenTipo tokenVal;
```

```
  char *stringValor;
```

```
  int tokenNum;
```

```
  int tokenProxPos;
```

```
} TokenRegistro[MAX_REG]
```

- Exemplo: array de token ([TokenArray.c](#))

# Observações

- O sistema de varredura normalmente retorna o registro do *token*
- A varredura ocorre normalmente **sob o controle do analisador sintático** por meio de uma função que retorna o tipo do token, por exemplo  
*TokenTipo getToken( )*
  - Exemplo: [getToken](#)
- Na maioria das linguagens, o sistema de varredura precisa apenas gerar um token por vez – verificação à frente do símbolo; mas criaremos um *array* de tokens com *MAX\_REG* valores
- Cada registro possui um índice no *array*

# Observações

- A cadeia de caracteres de entrada do sistema normalmente é armazenada em um repositório ou fornecida, por exemplo, um arquivo .txt
- Exemplo: Lendo [código fonte](#)
- Além dos *tokens* definidos, a implementação pode usar *tokens* de controle, por exemplo ENDFILE , que marca um final de arquivo e ERRO, que marca um erro na varredura
- Na verificação dos identificadores, após ler todos os caracteres, checa-se se a sequência não se configura uma palavra reservada por busca linear (outras técnicas podem ser usadas para aumentar a eficiência em linguagens reais)

# Observações

O resultado do sistema de varredura é o *array* de *token*, que será usado para dar entrada no analisador sintático

# Gramática da linguagem TINY - adaptada

*programa*  $\rightarrow$  **main** ( ) { *decl-sequência* }

*decl-sequência*  $\rightarrow$  *declaração* | *declaração* ; *declaração*

*declaração*  $\rightarrow$  *if-decl* | *atribuição-decl* | *while-decl*

*read-decl* | *print-decl*

*if-decl*  $\rightarrow$  **if** *exp* { *decl-sequência* } [ **else** { *decl-sequência* } ]

*atribuição-decl*  $\rightarrow$  **identificador** = *exp*

*while-decl*  $\rightarrow$  **while** *exp* { *decl-sequência* }

*read-decl*  $\rightarrow$  **read** **identificador**

# Gramática da linguagem TINY - adaptada

*print-decl*  $\rightarrow$  *print exp*

*exp*  $\rightarrow$  *simples-exp* [ *comparação-op* *simples-exp* ]

*comparação-op*  $\rightarrow$   $< | > | =$

*simples-exp*  $\rightarrow$  *termo* | *termo soma termo*

*soma*  $\rightarrow$   $+ | -$

*termo*  $\rightarrow$  *fator* | *fator mult fator*

*mult*  $\rightarrow$   $* | /$

*fator*  $\rightarrow$  ( *exp* ) | número | identificador



IBMEC.BR

 /IBMEC

 IBMEC

 @IBMEC\_OFICIAL

 @IBMEC

 **ibmec**