Compiladores

Aula 2

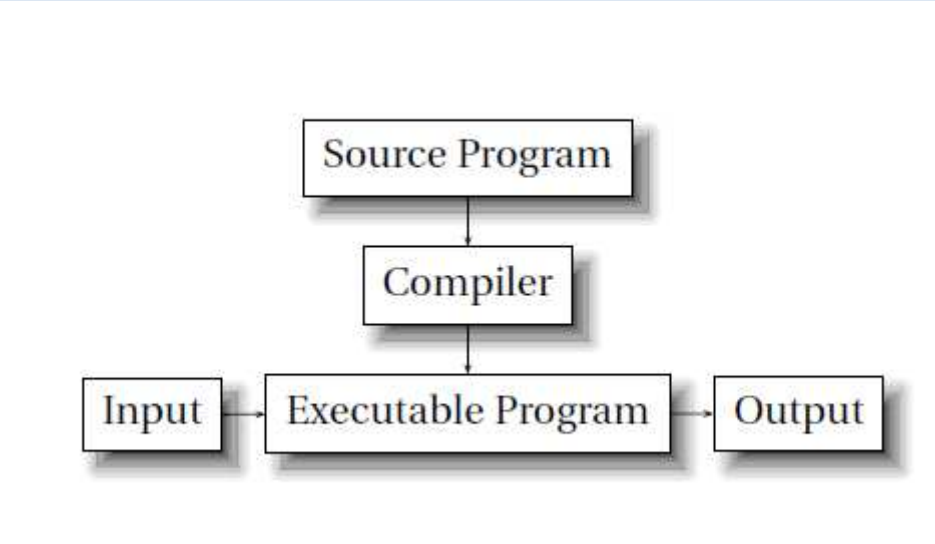
Celso Olivete Júnior

olivete@fct.unesp.br

Relembrando...

- Métodos de implementação

1. **Compilação:** Tradução do programa fonte → Linguagem máquina
   - ✓ Tradução lenta, execução rápida

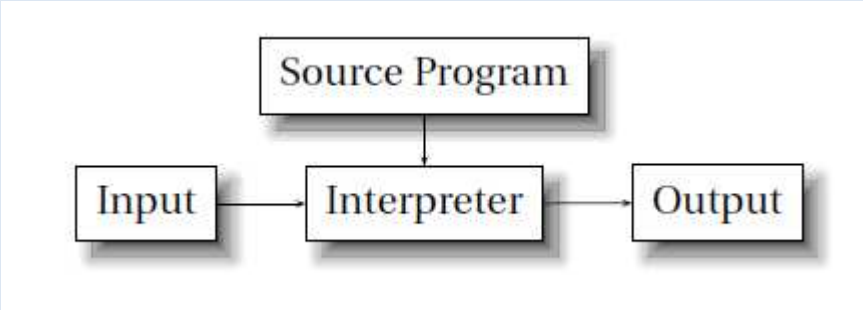


Relembrando...

- Métodos de implementação

  2. Interpretação: Traduz e executa

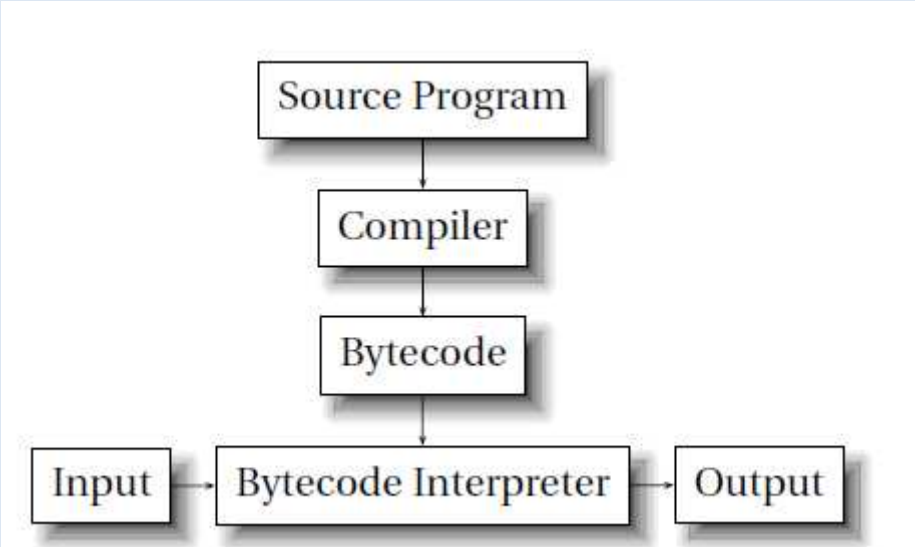
  - ✓ Consome muito tempo: de 10 a 100 vezes mais lento que um programa compilado



Relembrando...

- Métodos de implementação

3. Sistemas híbridos: Tradução do programa fonte → Linguagem intermediária (*bytecodes)* para facilitar a interpretação - **Máquina virtual** - (implementações iniciais de Java usavam esse conceito)



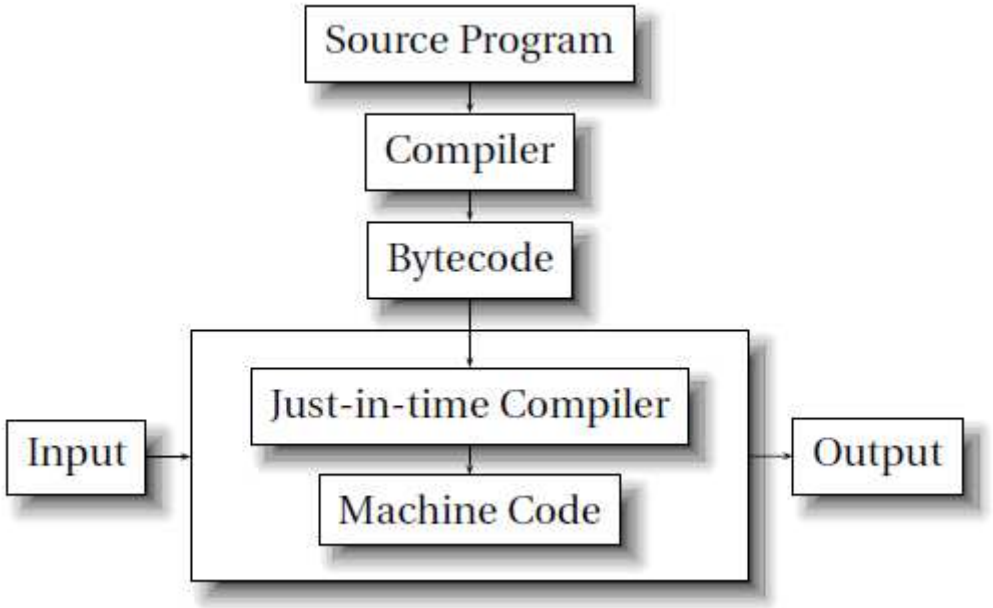
Relembrando...

- Métodos de implementação

3. Sistemas híbridos *Just – in – time*: Tradução do programa fonte → Linguagem intermediária (*bytecodes)* → compila os *bytecodes* em código de máquina nativa quando eles são chamados (tempo de execução) para executar na máquina nativa. Código é mantido para chamadas subsequentes

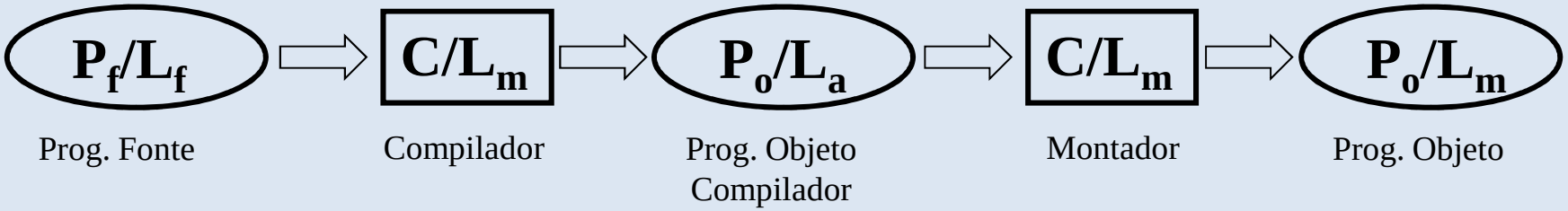
- ✓ Exemplo: Java

Nosso foco: processo de compilação



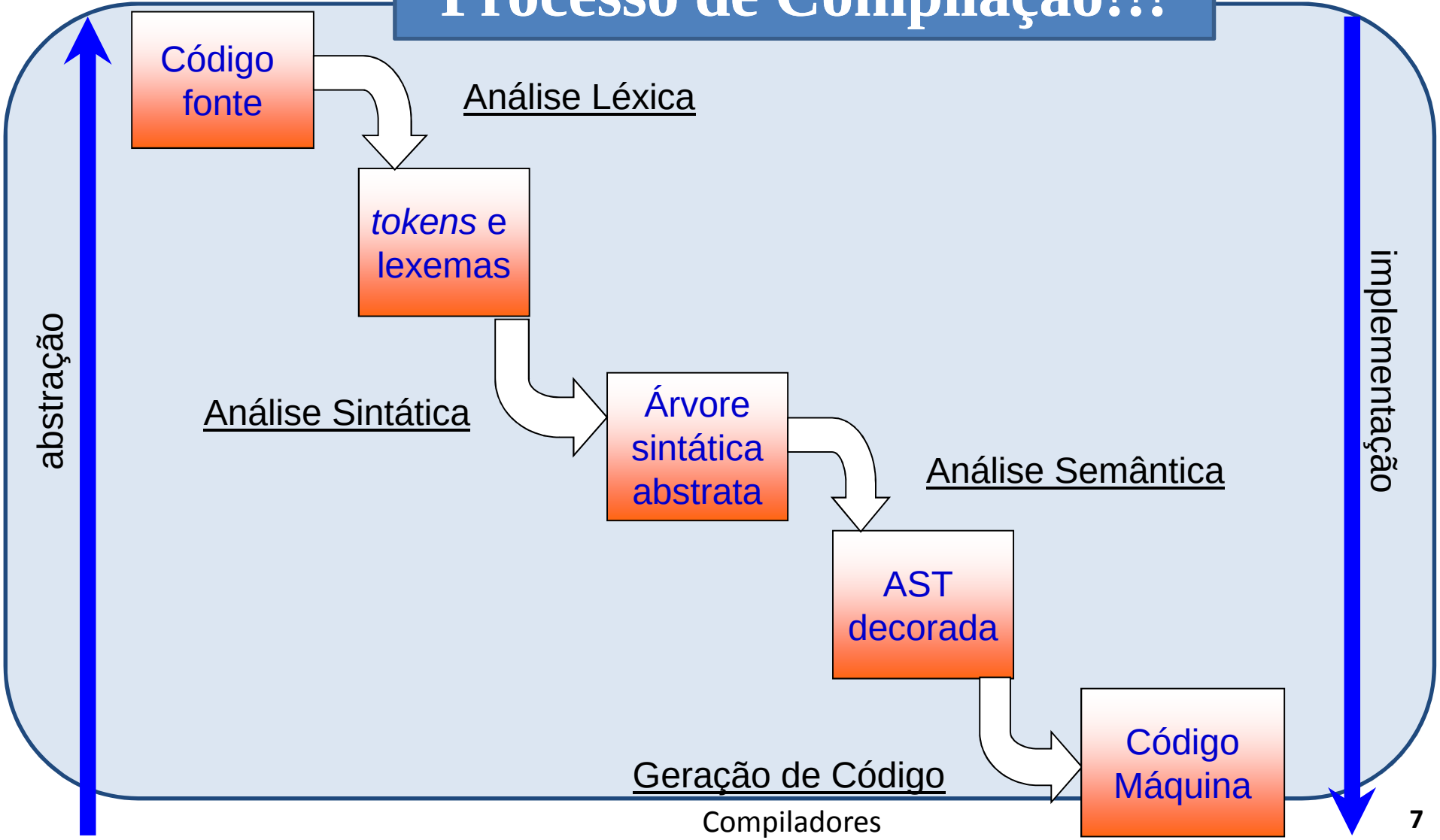
Relembrando...

- "Um compilador é um programa que transforma um outro programa escrito em uma linguagem de programação de alto nível qualquer em instruções que o computador é capaz de entender e executar, isto é, em código de máquina"



Relembrando...

Processo de Compilação!!!



Relembrando

- O compilador divide-se em duas etapas: análise e síntese
  - Etapa de **análise**:
    - ✓ Análise Léxica
      - ❖ Quebra a entrada em palavras conhecidas como *tokens*
    - ✓ Análise Sintática
      - ❖ Analisa a estrutura de frases do programa
    - ✓ Análise Semântica
      - ❖ Calcula o "significado" do programa

Relembrando

- O compilador divide-se em duas etapas: análise e síntese
  - Etapa de **síntese**:
    - ✓ Geração de código intermediário
    - ✓ Otimização do código
    - ✓ Geração do código objeto

Relembrando

- O compilador tem a responsabilidade de reportar erros
  - Etapa de **análise**:
    - ✓ Análise Léxica
    - ✓ Análise Sintática
    - ✓ Análise Semântica

Análise léxica (AL)

- **Erros léxicos:** O AL tem uma visão muito localizada no programa-fonte.

- Exemplo: fi (a > b) then
  - O AL não consegue dizer que fi é a palavra reservada if mal escrita desde que fi é um identificador válido
  - O AL devolve o código de identificador e deixa para as próximas fases identificar os erros

Análise léxica (AL)

- **Erros léxicos: exemplos**

- **Reais:** há um limite para o número de casas decimais
- ***Strings*:** o token **`aaaaaaaaaaaaaaaaaa ...** não fecha antes do tamanho máximo
  - é exemplo de má formação de *string*
  - há um limite para o tamanho da *string*
  - Se ferir o limite há erro

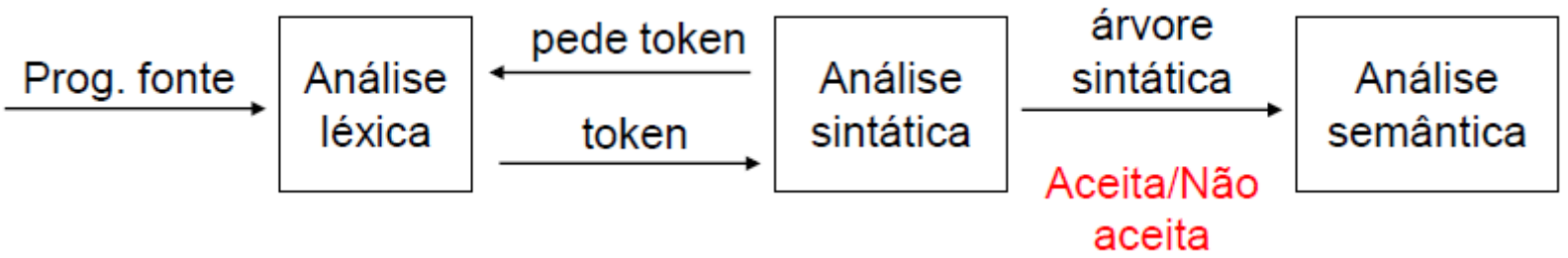
Análise léxica (AL)

❑ Outros erros léxicos

- ❑ Tamanho de identificadores
  - ➢ Geralmente, a linguagens aceitam até um tamanho de diferenciação e descartam o resto sem indicar erro
- ❑ Fim de arquivo inesperado
  - ➢ ocorre quando se abre comentário e não se fecha, por exemplo.
- ❑ & é um símbolo não pertencente ao Vt (vocabulário terminal da linguagem – símbolos da linguagem)
  - ➢ erros de símbolos não pertencentes ao Vt

Análise sintática(AS)

- **A AS é o processo de determinar**
  - se uma cadeia de átomos (*tokens*), isto é, o programa já analisado pelo AL, pode ser gerado por uma **gramática**
  - seu objetivo é a **construção da árvore sintática** ou apenas **a decisão** se a cadeia fornecida é ou não uma sentença da gramática que define a linguagem



Análise sintática(AS)

- **Erros sintáticos:** a AS se preocupará com a estrutura (formação) do programa.

- Exemplo: fi (a > b) then
  - O AL não consegue dizer que fi é a palavra reservada if mal escrita desde que fi é um identificador válido
  - É na AS que este erro é detectado

Análise sintática(AS)

- **Erros sintáticos:** a AS se preocupará com a estrutura (formação) do programa.

- Exemplos:
  - esquecer de abrir e/ou fechar parênteses ou aspas
  - usar uma palavra reservada como variável
  - uso de = ao invés de := (pascal)
  - writelm ao invés de writeln (pascal)
  - Fro ao invés de For
  - ...

Análise sintática(AS)

- **Erros sintáticos:**

```
1. program prmax.
2. var
3. x, y : integer;
4. function max (i: integer, j: integer) : integer
5. { retorna o maior de i e j }
6. begin
7. if i > j then max := i;
8. else max X= j
9. end;
10. X
11. readln (x, y);
12. writelm (max (x, y))
13. end.
```

Análise semântica (ASem)

- **A ASem é o processo de**
  - Calcular o "significado" do programa
    - **Erros semânticos**
      - ✓ tipos inválidos - o programa dá um resultado, mas não é correto!
      - ✓ o computador não advinha o que eu quero
      - ✓ logo: o programa que escrevi não resolve o problema pretendido
      - ✓ os erros semânticos são os mais difíceis de corrigir

Análise semântica (ASem)

- **Funções da ASem**
  - Checa os tipos de cada expressão
  - Relaciona declarações de variáveis com seus usos
  - É caracterizado pela manipulação de tabelas de símbolos
    - ✓ Mapeiam identificadores a seus tipos e localizações
  - Declarações geram inclusões nas tabelas de símbolos

Exercício

- Analise o código fonte (pascal) abaixo e identifique/classifique os erros em: léxicos, sintáticos e semânticos.

```
...
int I2, A@;
I2=0;
while I2>=0; do
    I:=I+1;
I2=´a´;
  2A=I2$5;
...
```

Exercício

- Analise o código fonte (pascal) abaixo e identifique/classifique os erros em: léxicos, sintáticos e semânticos.

```
...
int I2, A@;
I2=0;
while I2>=0; do
    I:=I+1;
I2='a';
  2A=I2$5;
```

léxico

sintático

semântico

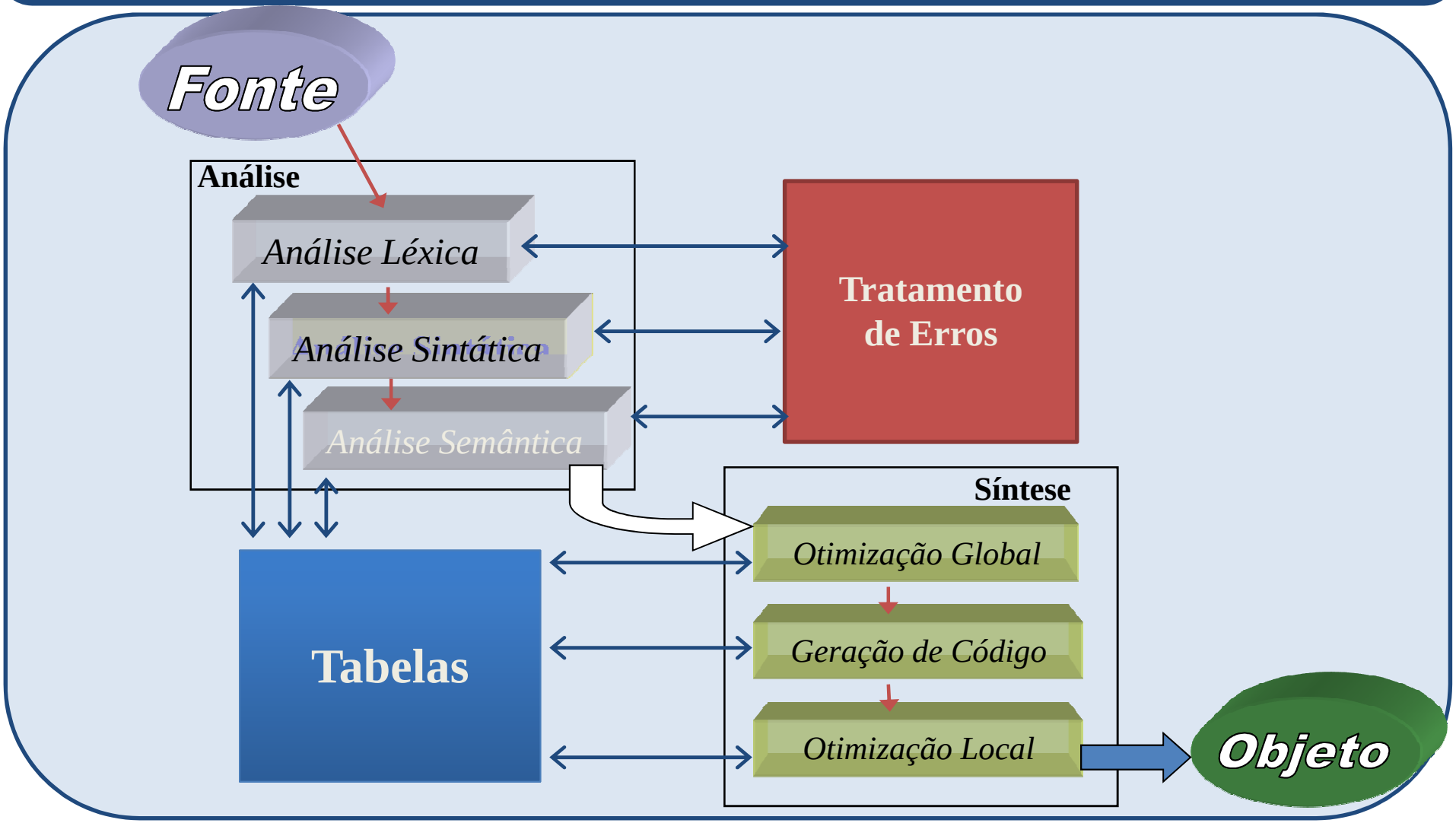
léxico

Reconhecimento de erros sintáticos

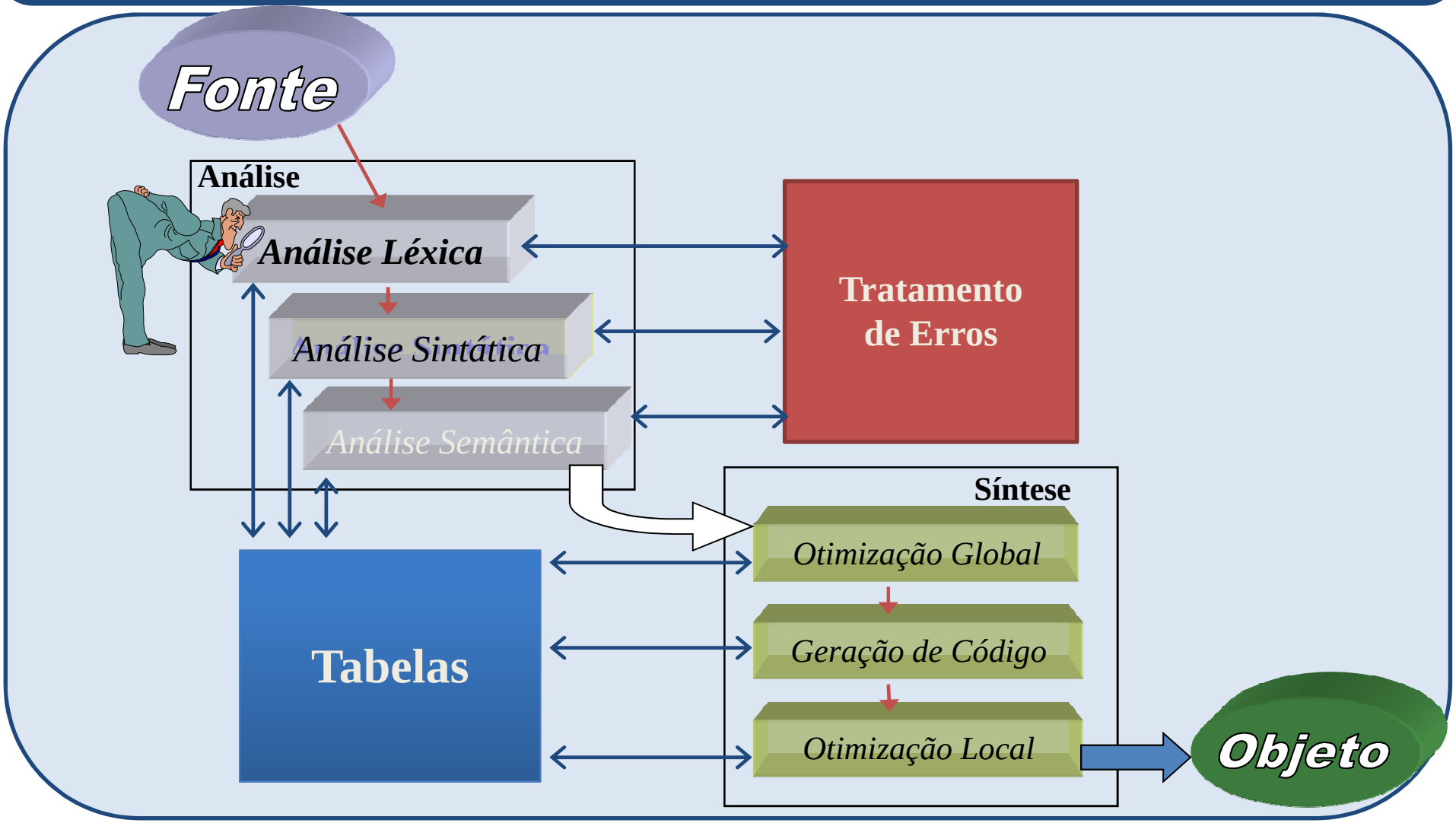
Utiliza-se Gramáticas Livre de Contexto

```
<expressão> ::= <expressão simples> [<relação> <expressão simples>]
<relação> ::= =| <>| <| <=| >=| >
<expressão simples> ::= [+| -] <termo> {(+| -| or) <termo>}
<termo> ::= <fator> {(*| div| and) <fator> }
<fator> ::=    <variavel>
               | <número>
               | (<expressão> )
               | not<fator>
```

Arquitetura básica de um compilador



Arquitetura básica de um compilador

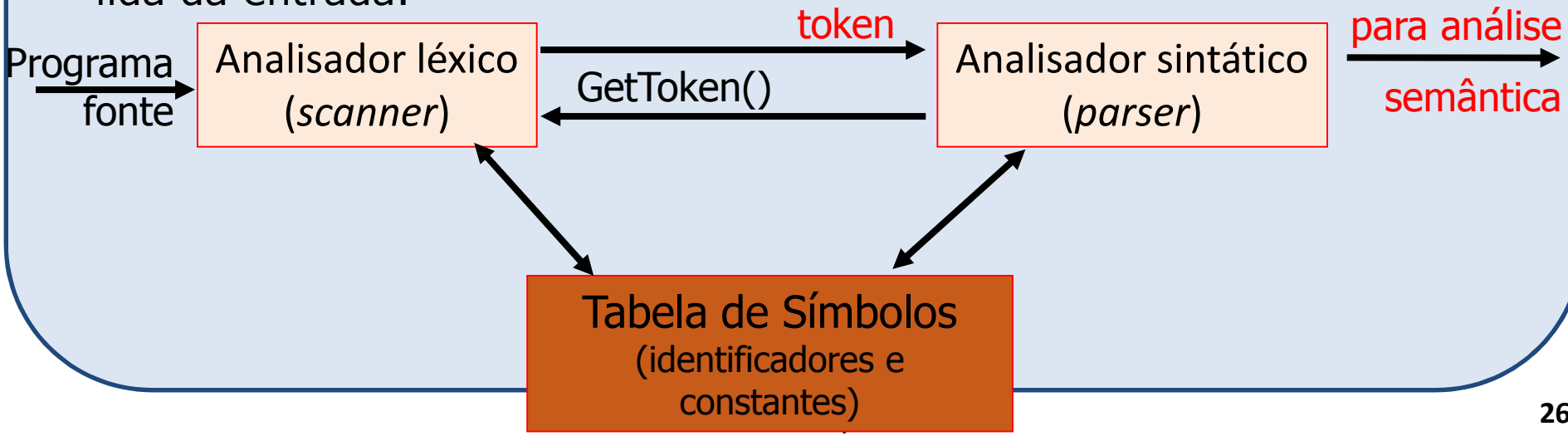


Na aula de hoje...

- **Análise léxica**
  - **Objetivos**
  - **Estratégias**
  - **Reconhecedores**

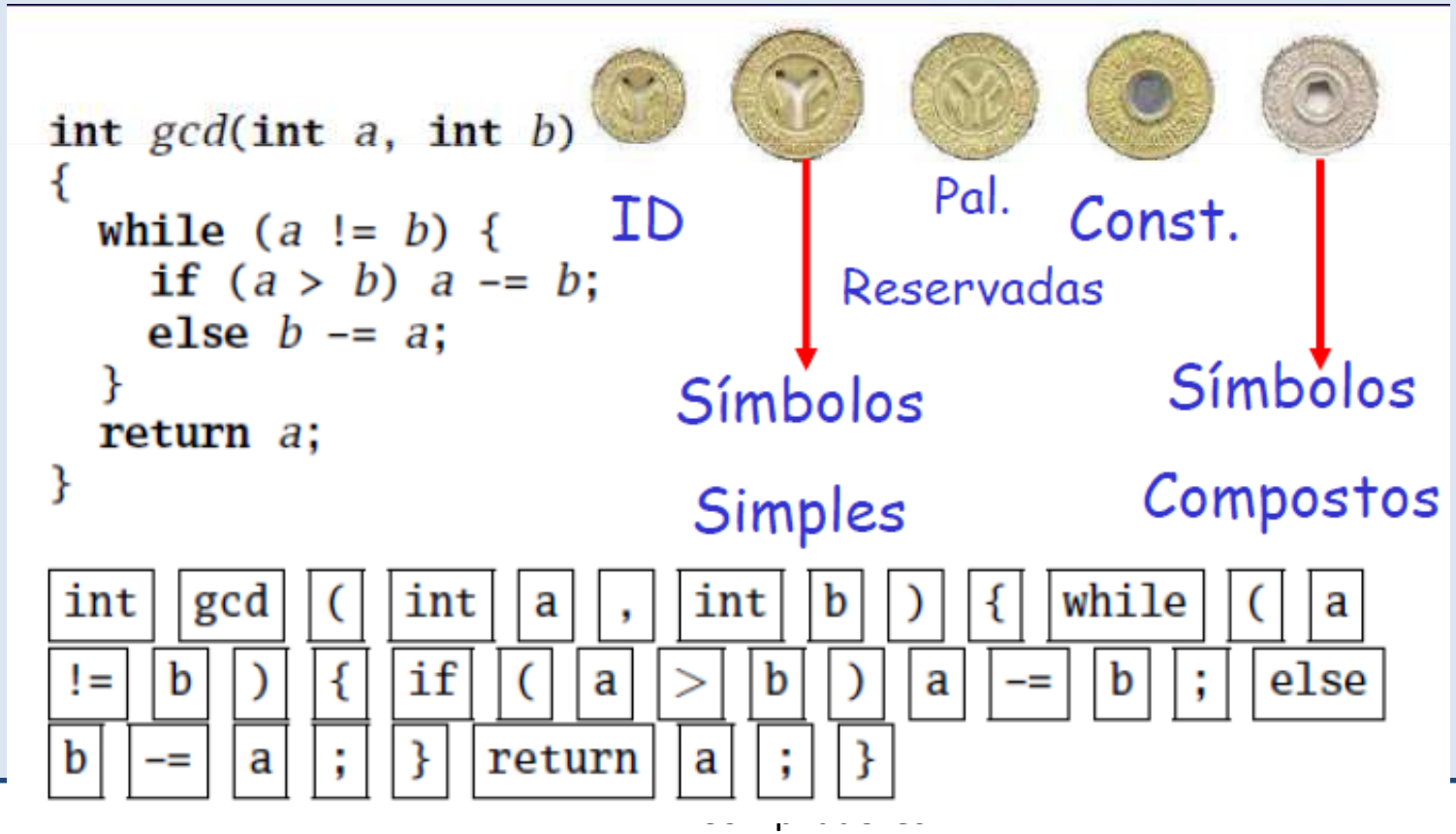
Análise léxica

- Seu objetivo é analisar a entrada dada (programa fonte) e dividi-la em sequências considerando os ***tokens*** da linguagem, definidos por expressões regulares. Cada ***token*** é normalmente formado por seu tipo (se é um operador lógico, um identificador, um número inteiro, etc.) e pelo seu valor e outros atributos. Este valor vai depender do tipo do *token* e normalmente corresponde à sequência de caracteres realmente lida da entrada.



Análise léxica

- A AL fornece *tokens* para o analisador sintático, desconsiderando espaços, comentários e quebra de linha



Analisador léxico

- **Tarefa principal**
  - ler o arquivo onde se encontra o programa-fonte e
  - produzir como saída uma sequência de *tokens* com seus respectivos códigos que o Analisador Sintático usará para validar regras da **gramática**

Extração e classificação dos *tokens*

- Classes de ***tokens*** mais comuns:
  - identificadores;
  - palavras reservadas;
  - números inteiros sem sinal;
  - números reais;
  - cadeias de caracteres;
  - sinais de pontuação e de operação;
  - caracteres especiais;
  - símbolos compostos de dois ou mais caracteres especiais;
  - comentários;
  - etc.

Eliminação de comentários

- O analisador léxico desconsidera o trecho do código fonte que encontra-se entre delimitadores de comentários.
- Além disso, ele desconsidera espaços em branco colocados pelos programadores a fim de melhorar a legibilidade do código fonte (endentação).

Analisador léxico

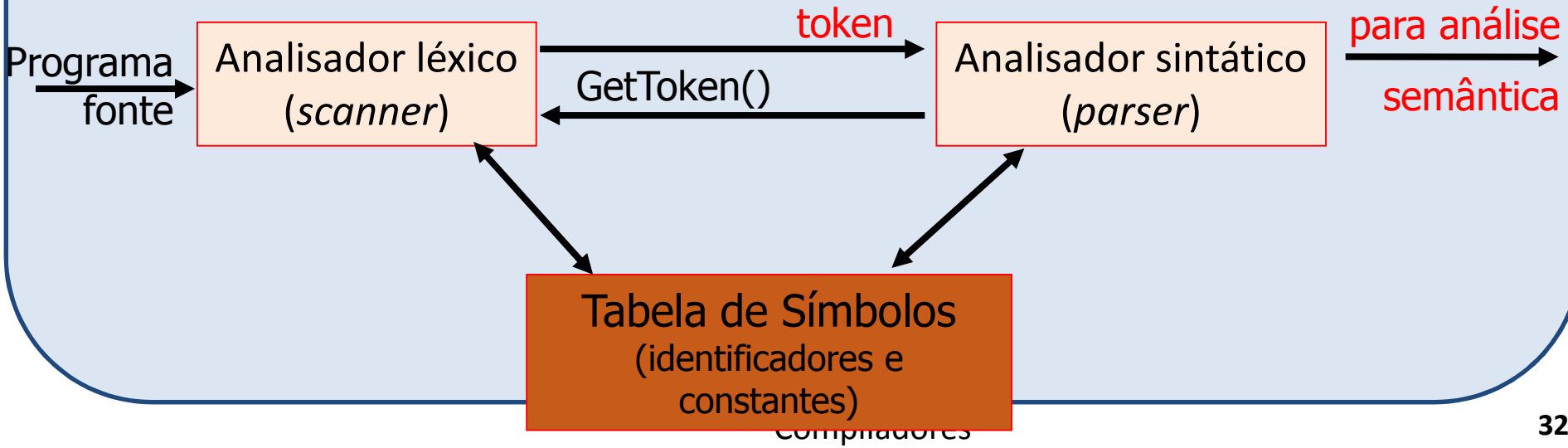
- Exemplos de *tokens* que podem ser reconhecidos em uma linguagem de programação como C

palavras reservadas	if else while do
identificadores	
operadores relacionais	< > <= >= == !=
operadores aritméticos	+ * / -
operadores lógicos	& | ! ...
operador de atribuição	=
delimitadores	; ,
caracteres especiais	( ) [ ] { }

Analisador léxico

- A interação é comumente implementada fazendo o AL como
  - Uma subrotina ou co-rotina do Analisador Sintático (AS)

- Quando o AS ativa a sub ou co-rotina,
  - o AL lê caracteres do arquivo até que ele possa identificar o próximo *token* e o devolve com seu código



Analisador léxico

- Exemplo: funcionamento do AL

```
x := y * 2;
```

Lexema	Token
x	id
:=	simb_atrib
y	id
*	simb_mult
2	num
;	simb_pv

Analisador léxico
outro exemplo

```
program p;
var x: integer;
begin
x:=1;
while (x<3) do
x:=x+1;
end.
```

O token integer em PASCAL é um identificador pré-definido, assim como outros tipos pré-definidos real, boolean, char, e também write, read, true e false

Lexema	Token
program	simb_program
p	id
;	simb_pv
var	simb_var
x	id
:	simb_dp
integer	p_res_integer
;	simb_pv
begin	simb_begin
x	id
:=	simb_atrib
1	num
;	simb_pv
while	simb_while
(	simb_apar
x	id
<	simb_menor
3	num
)	simb_fpar
do	simb_do
x	id
:=	simb_atrib
x	id
+	simb_mais
1	num
;	simb_pv
end	simb_end
.	simb_p

Analisador léxico

- O **analisador léxico** simplesmente varre a entrada (arquivo fonte) *em busca de padrões pertencentes a uma linguagem*.
- **Tratamento de erros durante o processo da AL:**
  - Tamanho de identificador maior que o permitido na especificação da linguagem
  - Fim de arquivo inesperado: comentário não fechado
  - Símbolo não pertence a linguagem: @
  - Inteiros/real longos: 1231234124132423 4
  - String mal formada: ‘a, ‘olá mundo
  - Número mal formado: 2.a4
- São limitados os erros detectáveis nessa etapa
  - Visão local do programa-fonte, sem contexto

    fi (a>b) then..

Analisador léxico
especificação precisa dos *tokens*

- Devemos usar notações formais para especificar a estrutura precisa dos ***tokens*** para construir um AL sem erros.
  - Por exemplo, mesmo a definição simples de cadeias de caracteres pode ser definida erroneamente se nada for dito sobre os caracteres permitidos:

    <string> ::= ‘<caractere> { <caractere>}’

- **OBS**: EM EBNF, {α} é zero ou mais vezes α

O scanner - exemplo

- O "scanner" lê o programa fonte caractere por caractere, juntando-os em unidades atômicas chamadas itens léxicos ou lexema.
- Há 2 tipos de itens:
  - cadeias específicas tal como IF ou ' ; `
  - classes de cadeias tal como identificadores, números e rótulos.

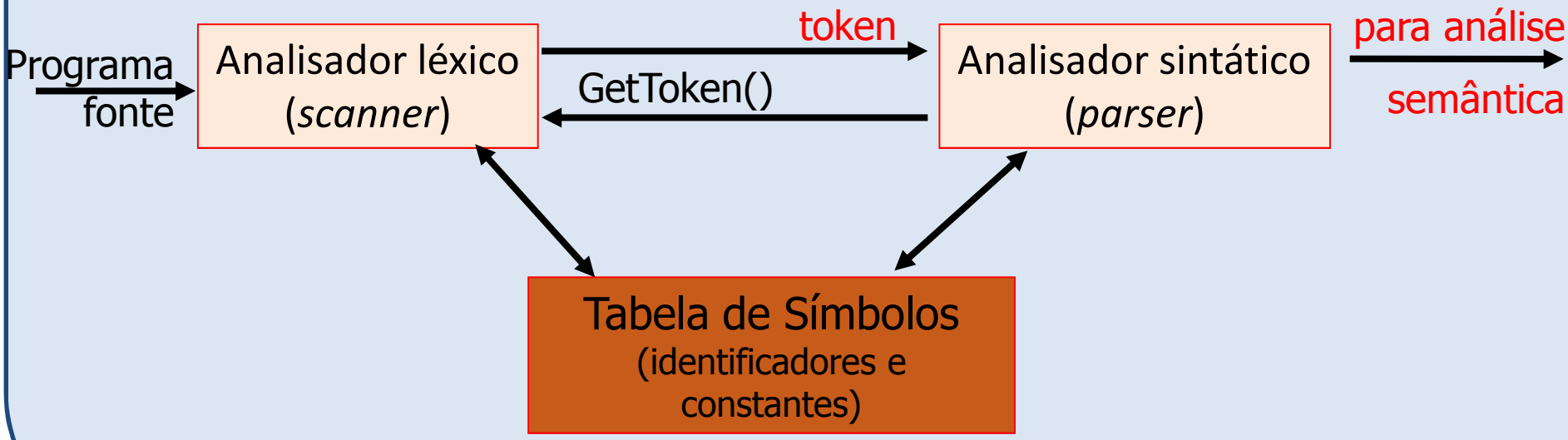
O scanner - exemplo

- Para tratar os 2 casos, vamos considerar um item como um par consistindo de 2 partes: o tipo de item e o valor do item. Por conveniência, um item consistindo de uma cadeia específica tal como ' ; ' , será tratado como tendo um tipo (própria cadeia), mas nenhum valor. Um item tal como um identificador, M, tem um tipo (identificador) e um valor consistindo da cadeia 'M'.
  - Recomenda-se apresentar a linha e colunas inicial/final de cada item
  - Exemplo

lexema (lido)	token (representa)	valor	linha	coluna inicial	coluna final	...
if	<palavra_reservada_IF>	-	Linha corresp.	Coluna inicial corresp.	Coluna final corresp.	

O scanner - exemplo

- A separação da **análise léxica** da **análise sintática** facilita o projeto e torna o compilador mais eficiente e portável.



Análise léxica – tratamento de erros

- Na ocorrência de erros o analisador léxico pode parar ou entrar em *loop* infinito. A **modalidade de pânico** pode ser usada para *recuperar erros léxicos* ignorando os caracteres inválidos até encontrar algum que pertença ao alfabeto ou o fim do arquivo.

- Outras formas de recuperar erros:
  1. remover caracteres estranhos
  2. inserir caracteres que faltam
  3. substituir caracteres incorretos por corretos
  4. trocar dois caracteres adjacentes

Análise léxica - Exemplo

- **G= {Vn, Vt, S, P}**
  - **Vn = {<exp>, <termo>, <fator>, <prim>}**
  - **Vt = { +, *, (,), ident, número}**
  - **S = {<exp>}**
  - **P: <exp>::= <termo> + <exp> | <termo>**
    <termo>::= <fator>*<termo> | <fator>
    <fator>::= <prim> <fator> | <prim>
    <prim>::= ident | (<exp>) | número

Análise léxica – Exemplo (continuação)

- Essa gramática aceita expressões aritméticas, com parênteses ou não, envolvendo as operações de soma e multiplicação entre operandos denominados genericamente de *ident* e *número.*
  - *Ident* pode ser qualquer cadeia formada pela combinação de letras e algarismos desde que começando por uma letra
    - ABC, A123, AB25D, X, I , I1, etc.
  - *Número* pode ser qualquer cadeia de algarismos. Por exemplo, são *ident* as seguintes cadeias:
    - 1, 123, 582, 123456780, 10, etc.

Análise léxica – Exemplo (continuação)

- Considere a sentença de entrada A:= A+BB1*32, o *scanner* executaria o seguinte:

Chamada	Lexema lida	token	valor	linha	...
1ª	A	ident	-	-	-
2ª	:=	:=	-	-	-
3ª	A	ident	-	-	-
4ª	+	+	-	-	-
5ª	BB1	ident	-	-	-
6ª	*	*	-	-	-
7ª	32	Número	-	-	-

Especificação de *tokens*

- *Tokens* são padrões que podem ser **especificados** através de **expressões regulares**.
- Um **alfabeto** determina o conjunto de caracteres válidos para a formação de cadeias, sentenças ou palavras.
- **Cadeias** são sequências finitas de caracteres. Algumas operações podem ser aplicadas a alfabetos para ajudar na definição de cadeias: concatenação, união e fechamento.

Especificação de *tokens*

- As **regras** para *definir expressões regulares* sobre um alfabeto são:

1. ε é a expressão regular para a cadeia vazia
2. a é a expressão regular para um símbolo do alfabeto {a}
3. se a e b são expressões regulares, também são expressões regulares:
   a) a | b
   b) a . b
   c) a*
   d) a^+

Especificação de *tokens*

- **Expressões regulares** podem receber um nome (**definição regular**), formando o *token* de um analisador léxico.

- Algumas convenções podem facilitar a formação de **definições regulares**

  1. Uma ou mais ocorrência ($^{+}$)
  2. Zero ou mais ocorrências (*)
  3. Zero ou uma ocorrência (?)
  4. Classe de caracteres [a-z]= a|b|...|z

Especificação de *tokens*

- São definições regulares
  - letra → [A-Z] | [a-z]
  - dígito → [0-9]
  - dígitos → dígito dígito*
  - identificador → letra[letra | dígito]*
  - fração_opc → .dígitos | ε
  - exp_opc → E[+ | - | ε]dígitos | ε
  - num → dígitos fração_opc exp_opc
  - delim → branco|tabulação|avanço de linha

Reconhecimento de *tokens*

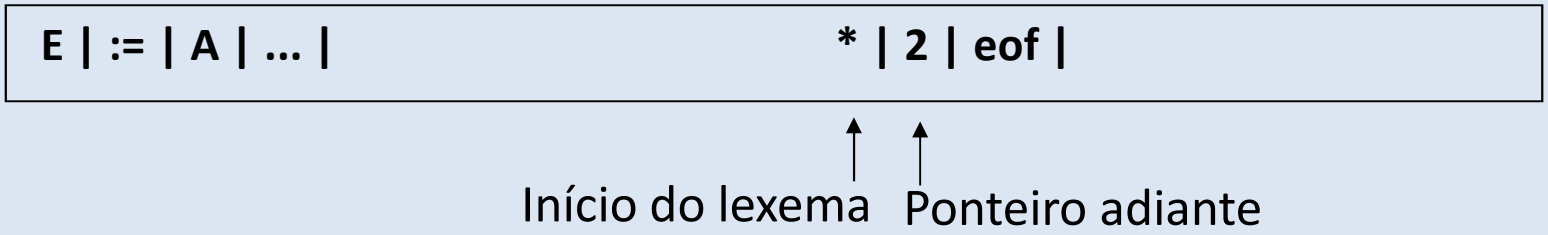
- *Tokens* podem ser reconhecidos por meio de autômatos finitos, sendo que o estado final dispara o reconhecimento de um *token* específico e/ou um procedimento específico (inserir na tabela de símbolo, por exemplo).

- Normalmente constrói-se um diagrama de transição para representar o reconhecimento de *tokens*.

Reconhecimento de *tokens*

❑ Diagramas de transição

- ❑ Mostram as ações tomadas pelo analisador léxico quando chamado pelo analisador sintático (*parser)*
- ❑ Considerando o buffer



- ➢ usamos um diagrama de transição a fim de controlar as informações a respeito dos caracteres que são examinado a medida que o apontador adiante

Reconhecimento de *tokens*

Diagramas de transição

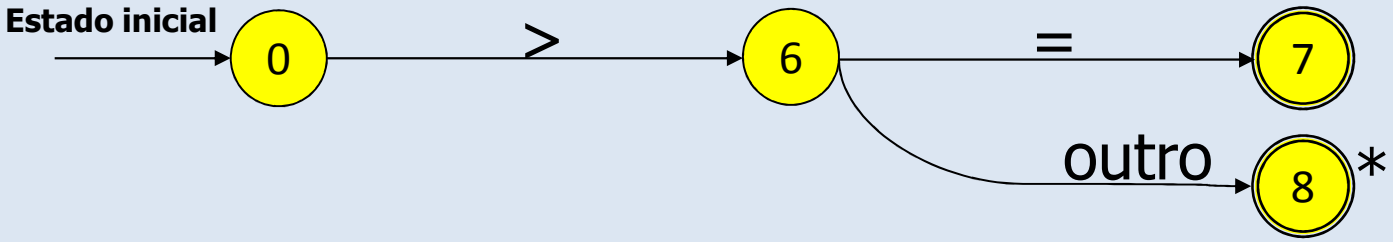
- Assumindo que os DT sejam determinísticos (o mesmo símbolo não pode figurar como rótulo em dois lados diferentes que deixem um estado)
- Os estados (representados por círculos) são conectados por setas, cujos rótulos indicam as possíveis sequências de caracteres
- O rótulo outro indica qualquer caracter que não apareça como esperado em um dado estado

Reconhecimento de *tokens*

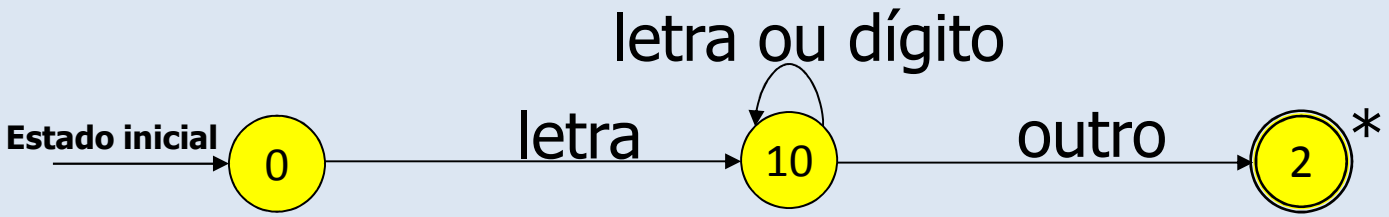
Diagramas de transição

- Estados podem ter ações associadas, que são executadas quando atingidos pelo fluxo de controle
- Círculos duplos indicam a aceitação de uma cadeia
- Asterisco indica o retrocesso do ponteiro

Exemplo para > e >=



Exemplo para identificadores



Reconhecimento de *tokens*

- Como são reconhecidos os identificadores e as palavras reservadas ?
- Como um compilador sabe o que é uma palavra reservada ?
- Há linguagens que permitem usar palavras reservadas como identificadores. Normalmente isto não acontece, mas o reconhecimento de identificadores e palavras reservadas é idêntico. É a tabela de símbolos que trata de identificar as palavras reservadas.

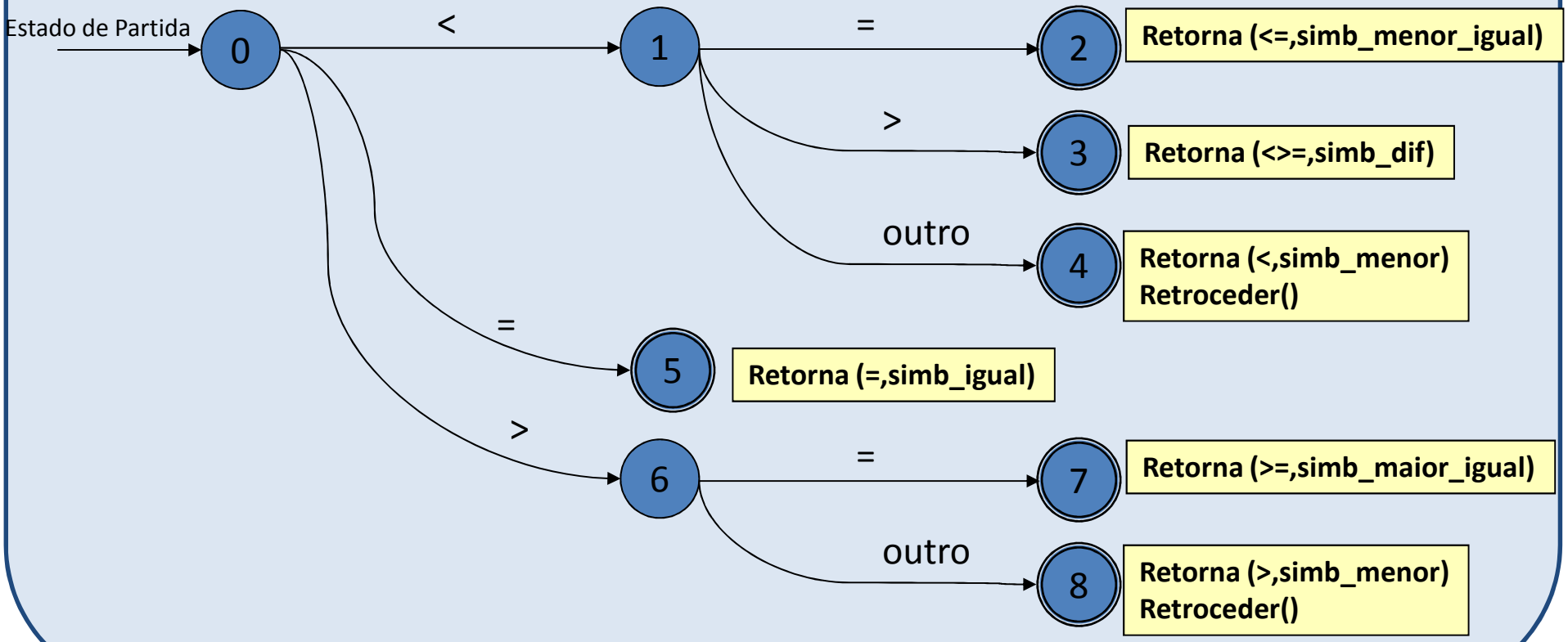
Reconhecimento de *tokens*

- Em geral a tabela de símbolos é *inicializada* com o registro das **palavras reservadas da linguagem**.

- O compilador sempre insere identificadores na tabela de símbolo? Isto é necessário?
  - Não, os identificadores são armazenados apenas uma vez, mas seus atributos podem ser alterados ao longo da análise de um programa.
    - int a;
    - a= 10;

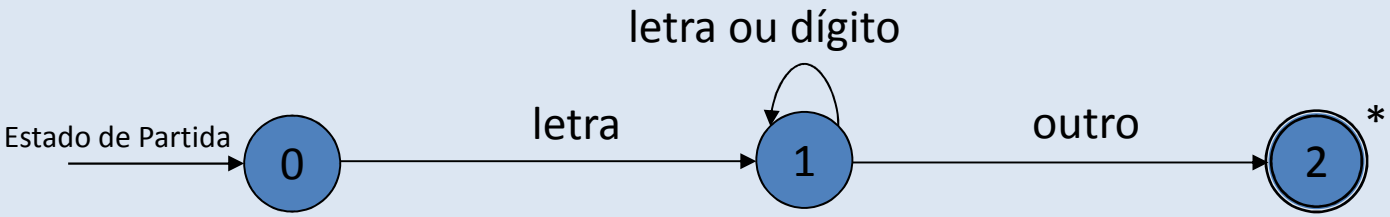
Reconhecimento de *tokens*

- **Exemplo: AF para reconhecer operadores relacionais**



Reconhecimento de *tokens*

- **Exemplo: AF para reconhecer identificadores**



Retorna obter_token(), tratar_token()

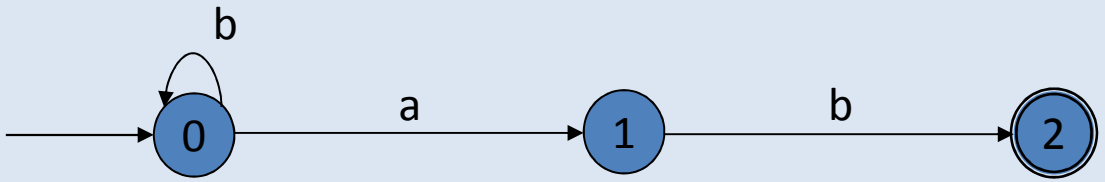
obter_token() ➔ retorna o token (if, then, variavel_x,...)

tratar_token() ➔ busca na tabela de palavras reservadas e classifica-o como sendo uma palavra reservada ou um identificador

Reconhecimento de *tokens*

- **Exemplo de implementação de um AF que reconhece b*ab**

```
c:=próximo_caractere()
se (c='b') então
        c:=próximo_caractere()
        enquanto (c='b') faça
                c:=próximo_caractere()
        se (c='a') então
                c:=próximo_caractere()
                se (c='b') e (acabou cadeia de entrada) então retornar "cadeia aceita"
                senão retornar "falhou"
        senão retornar "falhou"
senão se (c='a') então
                c:=próximo_caractere()
                se (c='b') e (acabou cadeia de entrada) então retornar "cadeia aceita"
                senão retornar "falhou"
senão retornar "falhou"
```



Projeto

- Primeira etapa: Analisador léxico para LALG

Na próxima aula

- Tratamento de erros léxicos para LALG