

# **Analizador Léxico**

## **Java/Ruby**



Francisco Laurindo Costa Júnior  
David Victor Cavalcante

# Visão geral

*A análise léxica é o processo de analisar a entrada de linhas de caracteres (tal como o código-fonte de um programa de computador) e produzir uma seqüência de símbolos chamados “símbolos léxicos” (lexical tokens), ou somente “símbolos” (tokens), que podem ser manipulados mais facilmente por um parser (leitor de saída). A Análise Léxica é a forma de verificar determinado alfabeto, quando analisamos uma palavra podemos definir através da análise léxica se existe ou não algum caracter que não faz parte do nosso alfabeto, ou um alfabeto inventado por nós.*

# **1. O papel do Analisador Léxico**

- a. Tokens, Padrões e Lexemas
- b. Erros léxicos

# **2. Especificação de tokens**

- a. Expressões regulares
- b. Definições regulares

# **3. Reconhecimento de tokens**

- a. Diagramas de transição
- b. Reconhecimento de palavras reservadas e identificadores.



## O papel do analisador léxico.

- Ler os caracteres do programa-fonte, agrupá-los em lexemas e produzir como saída uma sequência de tokens para cada lexema no programa-fonte.
- Enviar o fluxo de tokens para o analisador sintático.
- Registrar na tabela de símbolos informações sobre os identificadores encontrados.
- Eliminar comentários e espaços em branco
- Correlacionar mensagens de erros geradas pelo compilador com o programa-fonte.

programa-fonte



Analizador  
Léxico

-----token----->

<--getNextToken---

para análise  
semântica



Analizador  
Sintático




Tabela de  
Símbolos





## **Tokens, Padrões e Lexemas**

- Um **TOKEN** é um par consistindo de um nome e um valor de atributo(opcional). O nome do token é um símbolo abstrato que representa o tipo de unidade léxica.(Esperado como entrada pelo Analisador Sintático)
- O **PADRÃO** é uma descrição da forma que os lexemas de um token podem assumir. Pode ser definido por uma expressão regular.
- Um **LEXEMA** é uma sequência de caracteres do programa-fonte que casa com o padrão para um token e é identificado pelo analisador léxico como uma instância desse token.



<u>TOKEN</u>	<u>Descrição Informal</u>	<u>LEXEMAS</u>
<b>if</b>	caracteres i, f	if
<b>else</b>	caracteres e, l, s, e	else
<b>comparação</b>	< ou > ou <= ou >= ou == ou !=	<=, !=
<b>id</b>	letra seguida por letra e dígito	pi, x1
<b>número</b>	qualquer constante numérica	3,14159 , -2
<b>literal</b>	sequência de caracteres entre “	“ hoje “



## **Erros Léxicos**

**É difícil para o analisador léxico saber, sem o auxílio de outros componentes, que existe um erro no código-fonte.**

**● Por exemplo : fi é um identificador ou um if incorreto ???  
O erro que pode ser identificado pelo analisador léxico é um caracter inválido no programa-fonte. O Tratamento para esse tipo de erro é o que chamamos de Modo de Pânico onde removermos os caracteres seguintes até encontrar um token bem formado.**



# Especificação de Tokens

## Expressões Regulares.

TOKEN	Padrão	LEXEMAS
<b>if</b>	if	if
<b>else</b>	else	else
<b>Comparação</b>	(<   >   <=   >=   ==   != )	<=, !=
<b>id</b>	(a b c .... z).(a b c .... z 0 1 2.... 9)*	pi, x1
<b>Número</b>	(- ).(0 1 2 ... 9)+[(,(0 1 2 ... 9)+  ε]	3,14159 , -2
<b>literal</b>	"(a b c .... z 0 1 .... 9   _)*"	" hoje "



# Especificações de tokens

## Definições regulares

letra  $\rightarrow$  a|b|c|.....|z|A|B|C|...|Z

digito  $\rightarrow$  0 | 1 | 2 | ... | 9

id  $\rightarrow$  letra(letra|digito)\*



# Expressões Regulares

- Notação especial para definição de cadeias de uma linguagem
- Identificador Ruby
  - letra (letra|dígito)\*
  - Caractere | é igual a ou
  - \* significa zero ou mais instâncias
  - A justaposição de letras significa concatenação destas
  - Ex:
    - $a|b$  {a,b}
    - $(a|b)(a|b)$  {aa, ab, ba, bb}
    - $a^*$  {ε, a, aa, aaa, ...}
    - $(a|b)^*$
- Se duas expressões regulares denotam a mesma linguagem, dizemos que são equivalentes e representamos  $r=s$ . Ex:  $(a|b) = (b|a)$



## Definições Regulares

- Expressões regulares podem ser nomeadas e estes nomes podem ser utilizados para definição de novas expressões

Ex:

letra : A|B|...|Z|a|b|...|z

digito : 0|1|...|9

id : letra(letra|digito)\*

## Reconhecimento de Tokens

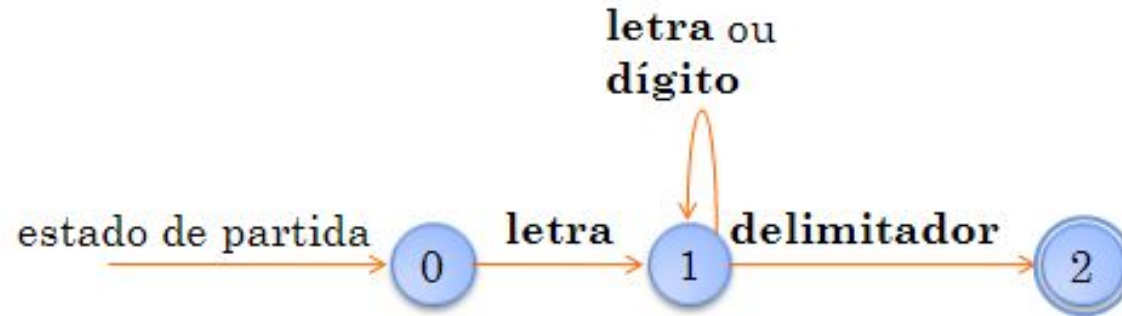
- **if** → início de uma condicional
- **then** → início da condicional verdadeira
- **else** → início da condicional false
- **< | <= | = | <> | > | >= |** → Operadores Lógicos
- **letra (letra | dígito)\*** → Identificador
- **dígito<sup>+</sup> (.dígito<sup>+</sup>)?(E(+ | -)?dígito<sup>+</sup>)?** → Número
- **branco | tabulação | avanço de linha** → delimitadores



## Diagrama de Transição

- Utilizado para determinar a seqüência de ações executadas pelo analisador léxico no processo de reconhecimento de um token;
- As posições no diagrama são representadas através de um círculo e são chamadas de **estado**;
- Os estados são conectados por setas, denominadas **lados**;
- Os lados são **rotulados** com caracteres que indicam as possíveis entrada que podem aparecer após o diagrama de estado ter atingido um dado estado;
- O rótulo **outro** se refere a qualquer caractere de entrada que não seja o indicado pelos demais rótulos que deixam o estado;
- Um círculo duplo determina um estado de aceitação;

# Técnicas de Reconhecimento de Palavras Chaves



A partir de uma entrada avaliar o token para verificar se é um identificador:


Um identificador inicia por letras e deve ser seguida por letras ou números:  
 $L (L \mid N)^*$



# **0 Analisador Léxico**







```
graph TD; A[Análise Léxica] --> B[Análise Sintática]; B --> C[Análise Semântica]; C --> D[Geração de Código Intermediário]; D --> E[Geração de Código Final];
```

Análise Léxica

Análise Sintática

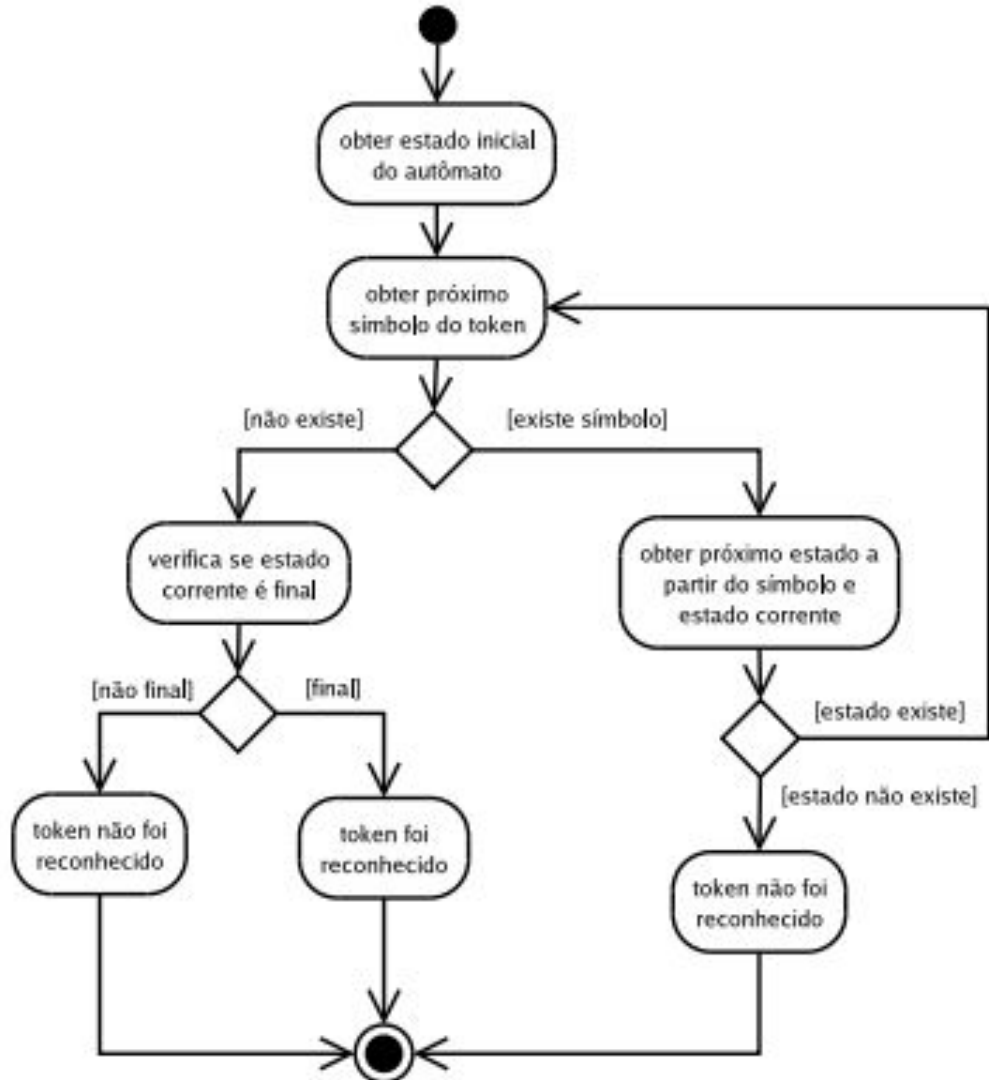
Análise Semântica

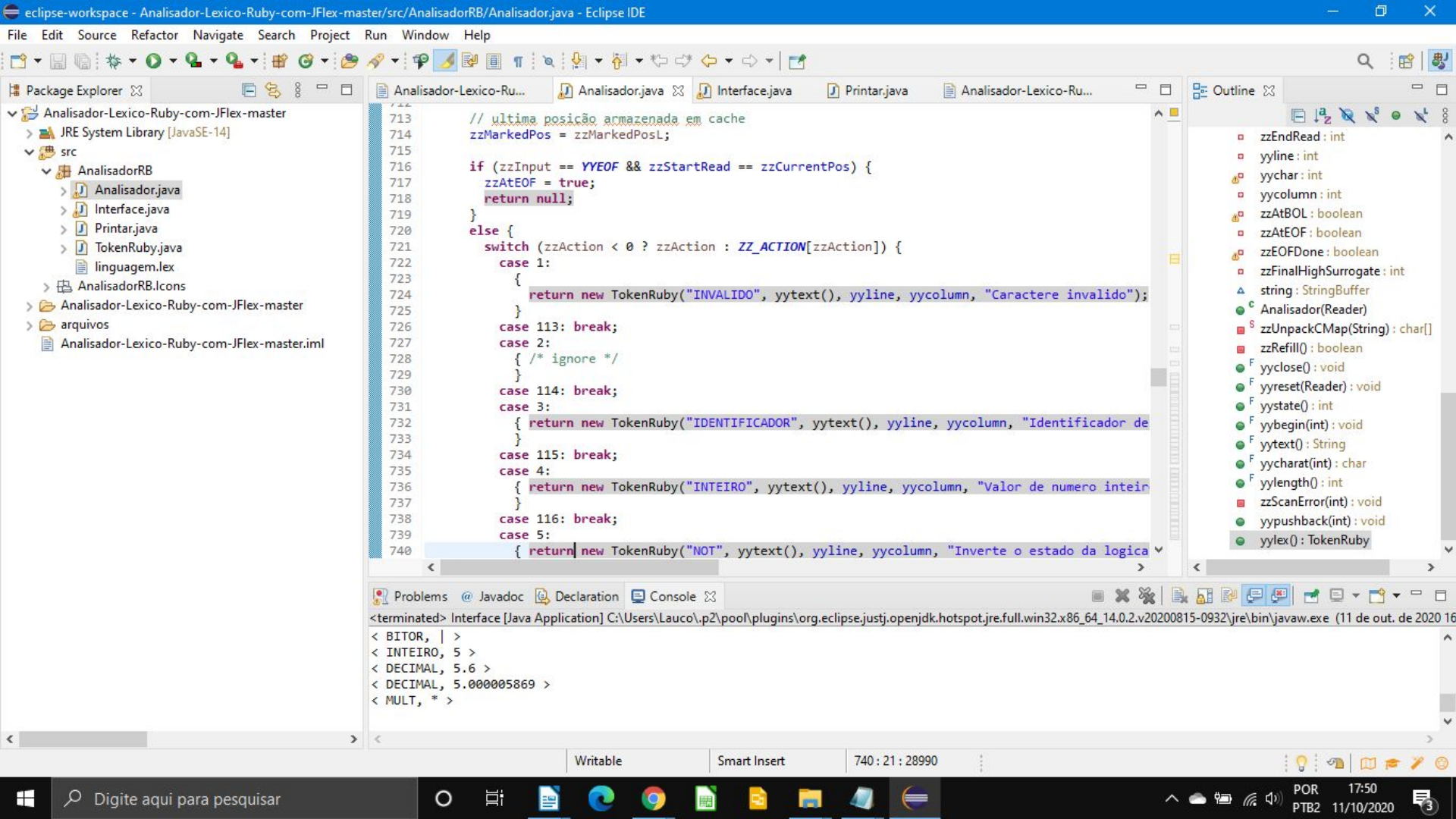
Geração de Código  
Intermediário

Geração de Código  
Final

*Front-End  
(Análise)*

*Back-End  
(Síntese)*





## Digite o Código Fonte

```
<>
%
==
+
-
*
/
//
\
|
5
5.6
5.000005869
*
```

Importar Arquivo.rb

Analisar

## Resultado da Análise

```
< MENOR, < >          Linha: 1 - Coluna: 1
Retorna true se o operando da esquerda for menor que o da direita: 5<10

-

< MAIOR, > >          Linha: 1 - Coluna: 2
Retorna true se o operando da esquerda for maior que o da direita: 10>5

-

< MOD, % >            Linha: 2 - Coluna: 1
Operador de modulo, retorna o resto de uma divisao

-

< EQEQ, == >          Linha: 3 - Coluna: 1
Verifica se dois operandos sao iguais: (a == a) retorna true, (a == b) retorna false

-

< PLUS, + >            Linha: 4 - Coluna: 1
Operador de soma

-

< SUB, - >             Linha: 5 - Coluna: 1
Operador de subtracao

-

< MULT, * >            Linha: 6 - Coluna: 1
Operador de multiplicacao
```

Salvar Análise

Limpar

Tutorial

**Obrigado!**