UNIVERSIDADE ESTADUAL PAULISTA

FACULDADE DE CIÊNCIAS E TECNOLOGIA

PROJETO COMPILADOR – ANALISADOR LÉXICO PARA LALG

RELATÓRIO – PARTE 1

COMPILADORES

PROF. DR. CELSO OLIVETE JÚNIOR

BRUNO SANTOS DE LIMA

LEANDRO UNGARI CAYRES

PRESIDENTE PRUDENTE

OUTUBRO - 2017

BRUNO SANTOS DE LIMA

LEANDRO UNGARI CAYRES

PROJETO COMPILADOR – ANALISADOR LÉXICO PARA LALG

RELATÓRIO

PARTE I

Relatório do projeto prático - parte I da disciplina de Compiladores, lecionada pelo docente Dr. Celso Olivete Júnior, no curso Bacharelado em Ciência da Computação – Departamento de Matemática e Computação da Faculdade de Ciências e Tecnologia (FCT Unesp – Presidente Prudente).

PRESIDENTE PRUDENTE

SETEMBRO – 2017

**SUMÁRIO**

[1 INTRODUÇÃO 5](#_Toc494632250)

[2 FUNÇÕES DO ANALISADOR LÉXICO 6](#_Toc494632251)

[3 CONJUNTO DE EXPRESSÕES REGULARES 7](#_Toc494632252)

[4 APLICAÇÃO DESENVOLVIDA 9](#_Toc494632253)

[4.1 Gerador JFlex 9](#_Toc494632254)

[4.2 Organização e interação entre as classes 10](#_Toc494632255)

[4.3 Testes realizados na ferramenta 10](#_Toc494632256)

[4.3.1 Teste 1: Programa correto 10](#_Toc494632257)

[4.3.2 Teste 2: Programa com comentários 13](#_Toc494632258)

[4.3.3 Teste 3: Programa incorreto 15](#_Toc494632259)

**LISTA DE FIGURAS**

[Figura 1 - Instantâneo da interface inicial da ferramenta 9](#_Toc494632343)

[Figura 2 - Exemplo 1: Programa correto1 11](#_Toc494632344)

[Figura 3 - Exemplo 1: Tabela de símbolos léxicos do programa correto1 11](#_Toc494632345)

[Figura 4 - Exemplo 2: Programa correto2 12](#_Toc494632346)

[Figura 5 - Exemplo 2: Tabela de símbolos léxicos do programa correto1 13](#_Toc494632347)

[Figura 6 - Exemplo 1: Programa com comentários corretos 14](#_Toc494632348)

[Figura 7 - Exemplo 2: Programa com comentário de múltiplas linhas incorreto 15](#_Toc494632349)

[Figura 8 - Exemplo 1 - Programa incorreto com caracteres inválidos 15](#_Toc494632350)

[Figura 9 - Exemplo 1 - Parte da tabela de símbolos do programa incorreto com caracteres inválidos 16](#_Toc494632351)

# 

# **1 INTRODUÇÃO**

Este trabalho da disciplina de Compiladores tem como objetivo a construção de uma aplicação, que realize a análise léxica para programas que utilizam a linguagem de programação LALG.

Este relatório está organizado em quatro seções, incluindo essa introdução. A Seção 2 explica brevemente as funções de um analisador léxico para um compilador, na Seção 3 são descritas as expressões regulares utilizadas e por fim a Seção 4 apresenta a aplicação, como ela foi implementada e um guia de utilização, além de mostrar alguns exemplos de teste da aplicação.

**2 FUNÇÕES DO ANALISADOR LÉXICO**

O analisador léxico é o processo de analisar uma entrada de um código fonte, realiza a varredura caractere a caractere, separando-os em lexemas, palavras pertencentes a linguagem e classificá-los em *tokens*.

Dentre as principais funções do analisador léxico estão:

* Extração dos lexemas do código fonte e classificação em *tokens*.
* Eliminação de espaços em branco.
* Eliminação de delimitadores e blocos de comentários.
* Identificação de símbolos não pertencentes a linguagem.

# 

# **3 CONJUNTO DE EXPRESSÕES REGULARES**

Para o desenvolvimento desse trabalho e consequentemente da aplicação foi necessário definir um conjunto de expressões regulares. As expressões regulares são importantes para ajudar na identificação dos Lexemas e Tokens processo fundamental da análise léxica, além de informar qual o alfabeto pertencente a linguagem.

A Tabela 1 apresenta os Tokens e a expressão regular utilizada para identifica-lo.

|  |  |
| --- | --- |
| Token | Expressão Regular |
| IDENTIFICADOR | ([\_|a-z|A-Z][a-z|A-Z|0-9]\*) |
| NUMERO\_INTEIRO | ([0-9]+{1,10}) |
| PALAVRA\_RESERVADA\_PROGRAM | (program) |
| PALAVRA\_RESERVADA\_BEGIN | (begin) |
| PALAVRA\_RESERVADA\_END | (end) |
| PALAVRA\_RESERVADA\_VAR | (var) |
| PALAVRA\_RESERVADA\_PROCEDURE | (procedure) |
| PALAVRA\_RESERVADA\_READ | (read) |
| PALAVRA\_RESERVADA\_WRITE | (write) |
| PALAVRA\_RESERVADA\_INT | (int) |
| PALAVRA\_RESERVADA\_BOOLEAN | (boolean) |
| PALAVRA\_RESERVADA\_IF | (if) |
| PALAVRA\_RESERVADA\_THEN | (then) |
| PALAVRA\_RESERVADA\_ELSE | (else) |
| PALAVRA\_RESERVADA\_WHILE | (while) |
| PALAVRA\_RESERVADA\_DO | (do) |
| VALOR\_LOGICO\_TRUE | (true) |
| VALOR\_LOGICO\_FALSE | (false) |
| SIMBOLO\_DOIS\_PONTOS | ( : ) |
| SIMBOLO\_PONTO\_VIRGULA | ( ; ) |
| SIMBOLO\_VIRGULA | ( , ) |
| SIMBOLO\_PONTO | ( . ) |
| SIMBOLO\_IGUAL | ( = ) |
| SIMBOLO\_DIFERENTE | ( <> ) |
| SIMBOLO\_MENOR | ( < ) |
| SIMBOLO\_MENOR\_IGUAL | (<= ) |
| SIMBOLO\_MAIOR | ( > ) |
| SIMBOLO\_MAIOR\_IGUAL | ( >= ) |
| OPERADOR\_ATRIBUICAO | ( := ) |
| OPERADOR\_ADICAO | ( + ) |
| OPERADOR\_SUBTRACAO | ( - ) |
| OPERADOR\_DIVISAO | (div) |
| OPERADOR\_MULTIPLICACAO | ( \* ) |
| OPERADOR\_AND | (and) |
| OPERADOR\_OR | (or) |
| OPERADOR\_NOT | (not) |
| PARENTESES\_ABRE | ( ( ) |
| PARENTESES\_FECHA | ( ) ) |
| COMENTARIO\_LINHA | ("\/\/"{COMENTARIO\_LINHA\_NAO\_FECHA}\*{COMENTARIO\_LINHA\_FECHA} ) |
| COMENTARIO\_LINHA\_FECHA | (\n) |
| COMENTARIO\_LINHA\_NAO\_FECHA | (^\n) |
| COMENTARIO\_MULTI | {COMENTARIO\_ABRE}{COMENTARIO\_CORPO\_MULTI}{COMENTARIO\_FECHA} |
| COMENTARIO\_ABRE | (\{) |
| COMENTARIO\_FECHA | (\}) |
| COMENTARIO\_NAO\_FECHA\_MULTI | (^}) |
| COMENTARIO\_CORPO\_MULTI | ({COMENTARIO\_NAO\_FECHA\_MULTI}\*) |

**Tabela 1** - Expressões Regulares utilizadas.

Com base nas expressões regulares definidas pode observar que o alfabeto A é:

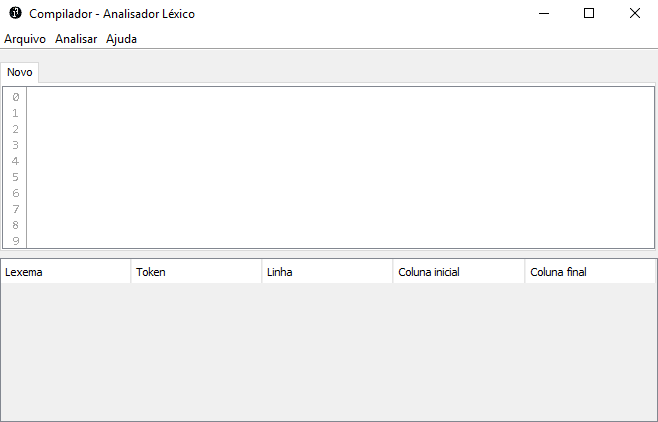
A = (“a-z”,”A-Z”,”0-9”,”+”,”-“,”\*”,”(“,”)”,”.”, ”,”,”;”,”:”,”>”,”<”,”=”,”{”,”}”,”/”)

Além disso um número inteiro pode ser representado com até 10 caracteres, mais que isso ele passa a ser outro número inteiro.

# 

# **4 APLICAÇÃO DESENVOLVIDA**

A aplicação foi desenvolvida na linguagem de programação JAVA utilizando a IDE NetBeans 8.2. Sendo assim é uma aplicação totalmente orientada á objetos. Para a interface grafia foi utilizado o pacote Swing do Java e para facilitar a identificação dos Lexemas e Tokens foi utilizado uma Biblioteca de Gerador denominada JFlex. Na Figura 1 é ilustrado a interface inicial da ferramenta.



**Figura 1** - Instantâneo da interface inicial da ferramenta

## **4.1 Gerador JFlex**

O JFlex é um conjunto de Bibliotecas que podem ser adicionadas a um projeto com o objetivo de facilitar a análise léxica. O JFlex é um gerador de analisador léxico e utiliza de expressões regulares previamente estabelecidas para realizar a analise léxica em uma entrada separando os Lexemas dando sua classificação em Tokens de acordo com as expressões regulares. O JFlex é baseado em soluções com autômatos finitos determinísticos. Para maiores informações consulte: <http://jflex.de/>

Neste trabalho o JFlex foi utilizado da seguinte forma: Primeiramente foi importado as bibliotecas do JFlex para o projeto, em seguida foi construído um arquivo chamado Lexer.lex que contém o conjunto de expressões regulares e um método para realizar a análise, esse arquivo Lexer.lex gera uma classe Java automaticamente Lexer.java que contém toda a implementação do gerador com base nas expressões regulares definidas.

## **4.2 Organização e interação entre as classes**

A aplicação contém um conjunto de classes organizadas em pacotes. A classe Arquivo.java é responsável por abrir e ler um arquivo texto que contém um código fonte que sofrerá uma análise léxica.

Com o código fonte lido através de um arquivo ou simplesmente digitado na área de texto pode-se executar a análise léxica, a classe AnalisadorLexico.java é a responsável por acionar o gerador lexer provido pelo JFlex e realizar a análise léxica e por último construir a tabela de símbolos léxicos. Para identificar cada um dos Tokens presentes na linguagem é utilizado uma classe enum Simbolo.java.

Uma classe Item.java representa cada lexema analisado, cada qual é inserido como um Item na tabela de símbolos léxicos e detalhado pelos atributos: lexema, *token*, número da linha, coluna inicial e coluna final.

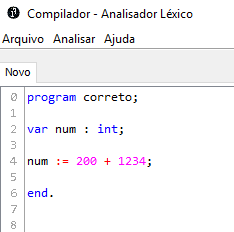
As demais classes são responsáveis pela construção da interface gráfica da aplicação.

## **4.3 Testes realizados na ferramenta**

A seguir são descritos alguns testes aplicados sobre a ferramenta. Os testes têm como objetivo mostrar o bom funcionamento da ferramenta e da sua capacidade em realizar a análise léxica e construir a tabela de símbolos léxicos.

### **4.3.1 Teste 1: Programa correto**

Dois testes foram realizados em programas corretamente escritos com todos os *tokens* pertencentes a linguagem e caracteres presentes no alfabeto. As Figuras 2 e 3 mostram os testes de um código simples podendo mostrar assim a tabela completa.

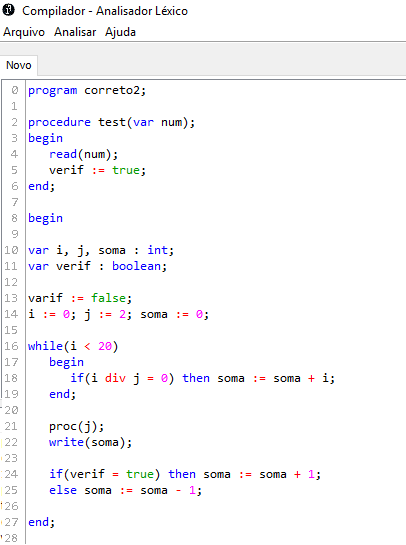


**Figura 2** - Exemplo 1: Programa correto1

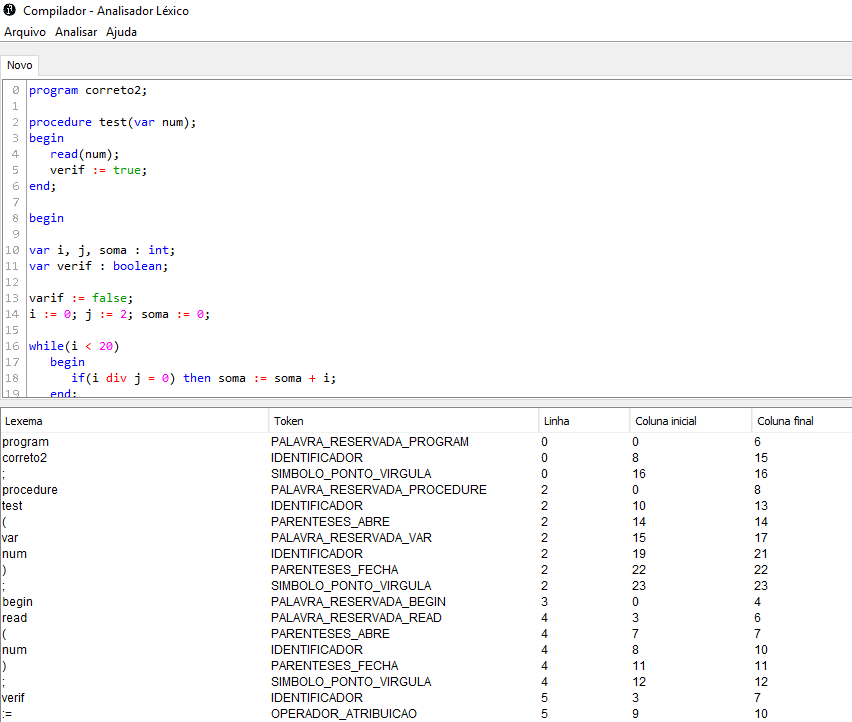


**Figura 3** - Exemplo 1: Tabela de símbolos léxicos do programa correto1

As Figuras 4 e 5 mostram um código mostrando apenas parte da tabela.



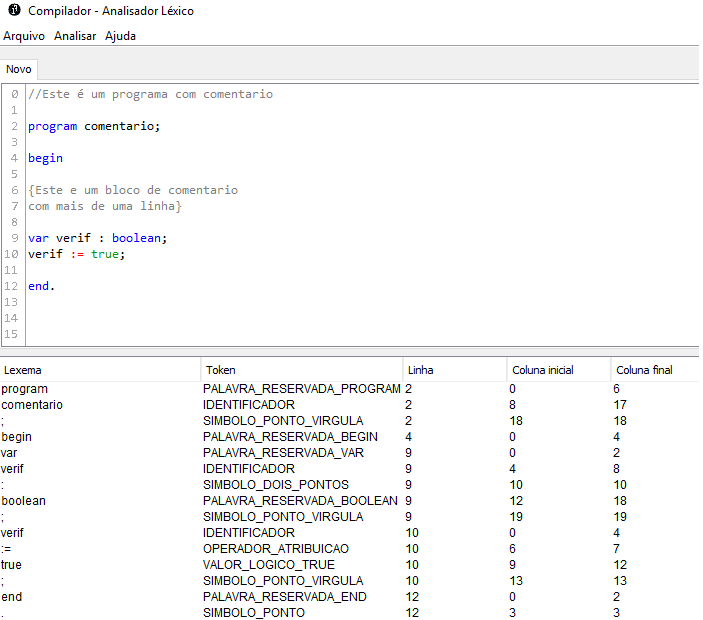
**Figura 4** - Exemplo 2: Programa correto2.



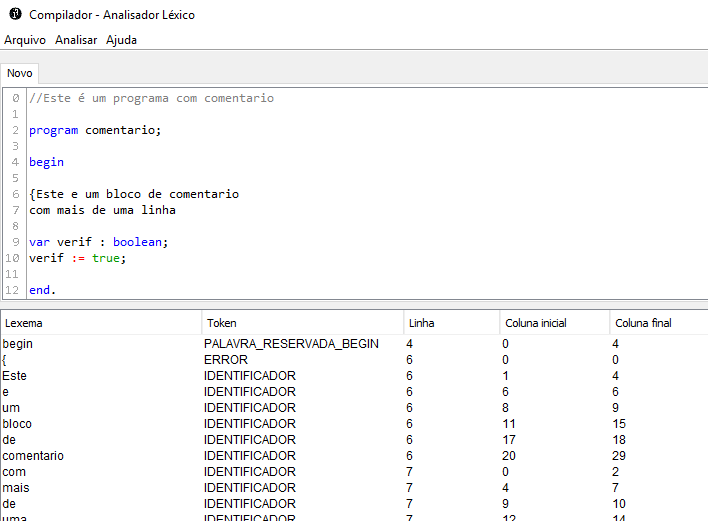
**Figura 5** - Exemplo 2: Tabela de símbolos léxicos do programa correto1.

### **4.3.2 Teste 2: Programa com comentários**

O segundo teste mostra os comentários tanto de uma única linha quando de múltiplas linhas. Observando a Figura 6, pode-se perceber que o analisador léxico cumpre sua função de ignorar comentários corretos. Contudo, na Figura 7 é ilustrado um caso onde o comentário de múltiplas linhas é aberto, porém não é fechado.



**Figura 6** - Exemplo 1: Programa com comentários corretos

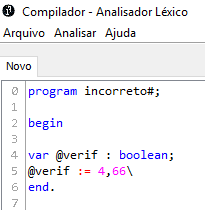


**Figura 7** - Exemplo 2: Programa com comentário de múltiplas linhas incorreto

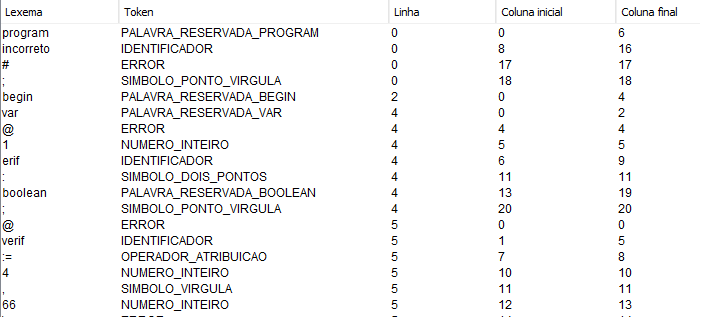
### 

### **4.3.3 Teste 3: Programa incorreto**

A seguir são apresentados programas incorretos contendo caracteres que não fazem parte da linguagem, apresentado na Figura 8, e na Figura 9 a respectiva tabela de símbolos léxicos.



**Figura 8** - Exemplo 1 - Programa incorreto com caracteres inválidos.



**Figura 9 -** Exemplo 1 - Parte da tabela de símbolos do programa incorreto com caracteres inválidos.