# Trabalho Prático 1 de Compiladores

## Análise Léxica

Douglas Rodrigues de Almeida
douglasralmeida@live.com

## 1. Introdução

O objetivo deste trabalho foi a criação de um analisador léxico que será usado no projeto de criação de um compilador durante a disciplina Compiladores I. O analisador foi implementado para a linguagem MLM, Mini Linguagem M, um subconjunto da linguagem Pascal, cuja especificação vem a seguir:

```
program ::= program identifier ";" decl_list compound_stmt

decl_list ::= decl_list " ;" decl
            | decl

decl ::= ident_list ":" type

ident_list ::= ident list " ," identifier
             | identifier

type ::= integer | real | boolean | char

compound stmt ::= begin stmt_list end

stmt_list ::= stmt list ";" stmt

            | stmt

stmt := assign_stmt | if_stmt | loop_stmt | read_stmt | write_stmt | compound_stmt

assign_stmt ::= identifier " :=" expr

if_stmt ::= if cond then stmt | if cond then stmt else stmt

cond ::= expr

loop_stmt ::= stmt_prefix do stmt_list stmt_suffix

stmt_prefix ::= while cond | E

stmt_suffix ::= until cond | end

read_stmt ::= read "(" iden_list ")"

write_stmt ::= write "(" expr_list ")"

expr_list ::= expr | expr_list " ," expr

expr ::= simple_expr | simple_expr RELOP simple_expr

simple_expr ::= term | simple_expr ADDOP term

term := factor_a | term MULOP factor_a

fator a := "-" factor | factor

factor := identifier | constant | " (" expr " )" | NOT factor

constant ::= integer_constant | real_constant | char_constant | boolean_constant

boolean_constant := false | true

unsigned_integer ::= digit digit*

sign ::= + | - | E

scale_factor := "E" sign unsigned_integer

unsigned_real ::= unsigned_integer ( E | "." digit*)( E | scale factor)

integer_constant ::= unsigned_integer

real_constant ::= unsigned_real
```

```
char_constant ::= "'" caractereASCII "'"
letter ::= A | B | ··· Z | a | b | ··· z
digit ::= 0 | 2 | 3 | 4 |5 | 6 | 7 | 8 | 9
identifier ::= letter ( letter | digit )*
RELOP ::= = | < | <= | > | >= | ! = | NOT
ADDOP ::= + | - | or
MULOP ::= ? | / | div | mod | and
```

## 2. Resumo do projeto

Um analisador léxico é utilizado para reconhecer símbolos terminais que foram utilizados no código-fonte de um programa. A identificação é feita por meio de *tokens*, um par formado pelo nome e um valor de atributo opcional, que caracteriza aquilo que foi reconhecido.

O analisador léxico foi implementado na linguagem Lex que é utilizada pela ferramenta JFlex (https://jflex.de/) para geração de analisadores prontos para serem compilados em Java. Essa ferramenta foi utilizada em detrimento ao JLex por ser compatível com as versões mais modernas do Java.

A ferramenta JFlex possui interface gráfica. Para gerar um código fonte Java do analisador bastar executar a ferramenta e registrar as instruções dos arquivos de entrada e saída nos campos específicos do aplicativo.

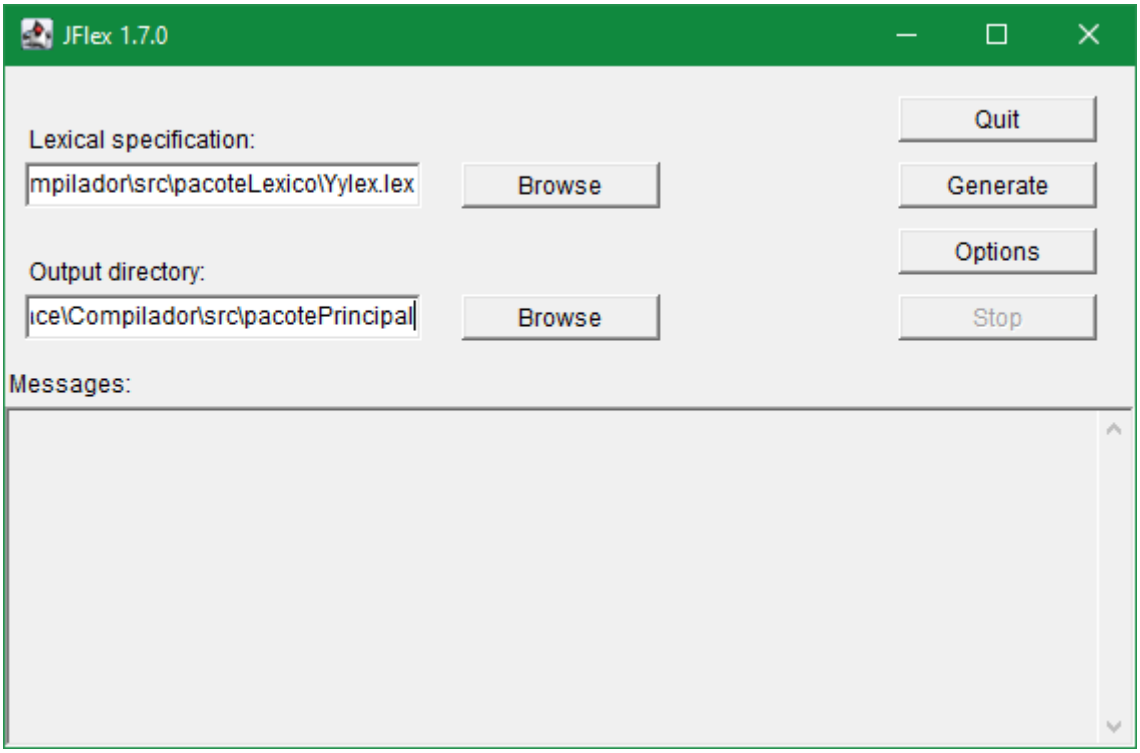


*Figura 1 - Ferramenta JFlex*

A especificação léxica está no arquivo Yylex.lex que se encontra na subpasta src/especificLex. A saída deve ser gerada na subpasta src/pacotePrincipal com o nome AnalisadorLexico.java.

O projeto foi criado usando a IDE Eclipse 2019.06 e OpenJDK 12 e pode ser compilado diretamente da IDE ou pela linha de comando:

```
java -jar D:\eclipse\plugins\org.eclipse.jdt.core_3.18.0.v20190522-0428.jar -d bin\
src\pacotePrincipal\
```

A linha de comando acima pode mudar dependendo da ferramenta de desenvolvimento utilizada e seu local de instalação.

Se compilado utilizando Java 12, a execução pela linha de comando será feita através da sintaxe:

```
java -p bin\ -m compilador/pacotePrincipal.Compilador <arquivoentrada>
```

O código-fonte está disponível no site do Github: https://github.com/douglasralmeida/comp1

## 3. Teste

O seguinte arquivo Teste.txt foi testado pelo analisador:

```
program OlaMundo;

b: boolean;
c: char;

begin
   b := true;
   c := 'X';
   write(c);
   if b = false then
     write('l');
end
```

O analisador deverá exibir os seguintes *tokens* na saída:

```
(program, )
(identifier, OlaMundo)
(identifier, b)
(boolean, )
(identifier, c)
(char, )
(begin, )
(identifier, b)
(constant, true)
(identifier, c)
(constant, X)
(write, )
(identifier, c)
(if, )
(identifier, b)
(RELOP, )
(constant, false)
(then, )
(write, )
(constant, l)
(end, )
```

## 4. Bibliografia


AHO, A. V. A.; SETH, R.; ULLMAN, J. D. **Compiladores. Princípios, Técnicas e Ferramentas**. Rio de Janeiro: LTC, 1995.

BERK, E. JLex: A lexical analyzer generator for Java. **JLex**, 2000. Disponivel em: <https://www.cs.princeton.edu/~appel/modern/java/JLex/current/manual.html>. Acesso em: 10 Setembro 2019.

KLEIN, G.; ROWE, S.; DECAMPS, R. JFlex User's Manual. **JFlex**, 2018. Disponivel em: <https://jflex.de/manual.html>. Acesso em: 10 Setembro 2019.

## 5. Código-fonte

Arquivo yylex.lex:

```
package pacotePrincipal;

%%

/* procedimentos */
%{
private void imprimir(String tipo, String valor) {
    System.out.println("(" + tipo + ", " + valor + ")");
}

%}

%class LexAnalisador
%type void

/* definicoes regulares */
delim=\r|\n|[\r\n]|\ |\t|\f
stoken={delim}+
letter=[a-zA-Z]
digit=[0-9]
identifier={letter}({letter}|{digit})*
unsigned_integer={digit}+
sign=[+-]?
scale_factor=E{sign}{unsigned_integer}
unsigned_real={unsigned_integer}(.{digit}*)?({scale_factor})?
integer_constant={unsigned_integer}
real_constant={unsigned_real}
char_constant=\'[^\r\n]\'
boolean_constant=false|true

%% /* regras de traducao */
{stoken}     { /* ignora */ }
<YYINITIAL> {
    /* operadores de relação */
    "="  |
    "<"  |
    "<=" |
    ">"  |
    ">=" |
    "!=" |
    "not"    { imprimir("RELOP", yytext()); }

    /* operadores de adição */
    "+"  |
    "-"  |
    "or"     { imprimir("ADDOP", yytext()); }

    /* operadores de multiplicação */
```

```
    "*"   |
    "/"   |
    "div" |
    "mod" |
    "and"    { imprimir("MULOP", yytext()); }

    /* palavras reservadas */
    "program" { imprimir("program", ""); }
    "integer" { imprimir("integer", ""); }
    "real"    { imprimir("real", ""); }
    "boolean" { imprimir("boolean", ""); }
    "char"    { imprimir("char", ""); }
    "begin"   { imprimir("begin", ""); }
    "end"     { imprimir("end", ""); }
    "if"      { imprimir("if", ""); }
    "then"    { imprimir("then", ""); }
    "else"    { imprimir("else", ""); }
    "do"      { imprimir("do", ""); }
    "while"   { imprimir("while", ""); }
    "until"   { imprimir("until", ""); }
    "read"    { imprimir("read", ""); }
    "write"   { imprimir("write", ""); }

    /* temporario */
    ":=" {}
    ";"  {}
    ":"  {}
    "("  {}
    ")"  {}
}

/* constantes */
{boolean_constant}  { imprimir("constant", yytext()); }
{integer_constant}  { imprimir("constant", yytext()); }
{real_constant}     { imprimir("constant", yytext()); }
{char_constant}     { imprimir("constant", yytext().substring(1, 2)); }

/* outros */
{identifier} { imprimir("identifier", yytext()); }
<<EOF>>      { System.exit(0); }
```

Arquivo Compilador.java:

```
package pacotePrincipal;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.StringReader;

public class Compilador {
```

```
    public static void main(String[] args) throws IOException {
        StringReader sr = null;

        if (args.length < 1) {
            System.out.println("A sintaxe do comando está incorreta.");
            System.out.println("Use: java -p bin\\ -m compilador/pacotePrincipal.Compi-
lador <arquivoentrada>");
            System.exit(1);
        }
        try {
            BufferedReader br = new BufferedReader(new FileReader(args[0]));
            StringBuilder sb = new StringBuilder();
            String line = br.readLine();
            while (line != null) {
                sb.append(line);
                sb.append(System.lineSeparator());
                line = br.readLine();
            }
            br.close();
            sr = new StringReader(sb.toString());
        } catch (IOException ex) {
            System.out.println("Ocorreu um erro ao carregar o arquivo.");
            System.exit(1);
        }
        try {
            AnalisadorLexico lex = new AnalisadorLexico(sr);
            lex.yylex();
        } catch (IOException ex) {
            System.out.println("Ocorreu um erro ao processar o arquivo.");
            System.exit(1);
        }
        System.exit(0);
    }
}
```