
# Teoria dos Compiladores - Analisador Léxico

UFJF

Matheus Gomes Luz Werneck (201835037)
Pedro Henrique Almeida Cardoso Reis (201835039)

Juiz de Fora, 16 de Abril de 2023

# Contents

**1 Introdução** **1**

**2 Instruções para o uso do projeto** **1**
2.1 Como rodar o projeto . . . . . 1

**3 Analisador Léxico** **1**
3.1 Expressões Regulares . . . . . 2
3.2 Processamento dos Tokens . . . . . 2

**4 Considerações Finais** **3**

# 1 Introdução

Este relatório do trabalho de Teoria dos Compiladores tem como objetivo explicar primeiramente como foi implementado o Analisar Léxico da linguagem *Lang*. O trabalho foi implementado em Java além de usar o JFlex como uma biblioteca para nos auxiliar no desenvolvimento do código.

# 2 Instruções para o uso do projeto

No projeto encontra-se três diferentes Schell Scripts para a compilação do mesmo.

- **execJflex.sh:** Este primeiro Shell Script é responsável por executar o JFlex passando o arquivo de especificação *LangSpec.jflex* como um parâmetro de entrada;
- **compileAndRun.sh:** O compileAndRun.sh é responsável por chamar o shellscript execJflex para gerar a classe LexicalAnalyser, e, após isso, compilar o projeto e executra o mesmo. É passado como parâmetro o nome do arquivo de texto a ser processado;
- **compileFolder.sh** Esse shell script executa os arquivos de testes que encontram-se em *testes/semantica/certo* e *testes/semantica/errado*. Após a execução desse script as saídas são gravadas em um arquivo .txt na pasta *resultados*.

## 2.1 Como rodar o projeto

- Certifique de ter instalado em seu computador o Java e o JDk.
- Para executar o programa, abra um terminal, e execute o script **compileAndRun.sh**. É obrigatório passar como parâmetro o nome do arquivo de entrada a ser processado, como por exemplo:

```
./compileAndRun.sh entrada1.txt
```

- Após digitar o comando, será impresso na tela token a token contido no arquivo texto, ou uma mensagem de erro em caso de erro.

# 3 Analisador Léxico

A primeira parte do trabalho de Teoria dos Compiladores é a entrega do Analisador Léxico. Como citado na introdução deste relatório, o trabalho foi implementado em Java, além de usar o JFlex como uma biblioteca para nos auxiliar no desenvolvimento do código. É com o JFlex que é gerado os autômatos, facilitando assim a nossa implementação. Esse trabalho possui como base quatro classes principais: *App.java;* $TOKEN_TYPE.java$; $Token.java$; $LangSpec.jflex$

***App.java:*** *É a classe "Main" do nosso trabalho. Ela recebe por parâmetro de linha de comando o path do txt de entrada. É responsável por chamar a classe* ***LexicalAnalyser.java*** *e pegar token a token do arquivo de entrada e imprimir na tela.*

***TOKENTYPE.java:*** *Nessa classe usamos um tipo especial de dados:* `enum`*. O enum nada mais é que um tipo de dados especial que permite que uma variável seja um conjunto de constantes predefinidas. Em outras palavras, é aqui onde definimos tudo aquilo que é considerado um Token.*

***Token.java:*** *Classe reponsável por guardar as informações referentes aos tokens, como o tipo do token, lexema e linha e coluna onde o mesmo está encontrado no arquivo de entrada.*

***LangSpec.jflex:*** *Classe gerada a partir da especificação contida no arquivo* ***LangSpec.jflex****. Essa classe contêm todas as funcionalidades do analisador léxico.*

## 3.1 Expressões Regulares

As expressões regulares foram definidas para ajudar a encontrar cada tipo de Token especificado na linguagem *Lang*, como mostra a figura abaixo:

```
endOfLine  = \r|\n|\r\n
whitespace     = {endOfLine} | [ \t\f]
int      = [:digit:]+
float        = [:digit:]*\.[:digit:]+
whiteSpaceChars = \\t | \\n | \\b | \\r
char         = '([:uppercase:] | [:lowercase:] | \\ | \\' | {whiteSpaceChars})'
bool= "true" | "false"

identifier = [:lowercase:]+ ( [:lowercase:]* [_]* [:uppercase:]* [:digit:]* ) *
lineComment = "--" (.)* {endOfLine}

reservedWord = "if" | "then" | "else" | "iterate" | "read" | "print" | "return"
typeName = [:uppercase:] [:lowercase:]+ ([:uppercase:] | [:lowercase:])+
```

Figure 1: Expressões Regulares

## 3.2 Processamento dos Tokens

O processamento dos Tokens é feito a partir do *<YYINITIAL>* como mostra a figura abaixo. É aqui que retornamos os tokens definidas na classe *TOKENTYPE*. Vale dizer também que a ordem que os tokens são definidos é importante. Por exemplo, para coletarmos o token == precisamos definir ele antes do token =. Nesse exemplo, se definirmos o token = antes do token == o analisador

coletará somente o token de atribuição, sendo que estamos realizando uma comparação de igualdade.

```
<YYINITIAL>{
    {whitespace}    {}
    {bool}       { return symbol(TOKEN_TYPE.BOOL);}
    "null"          { return symbol(TOKEN_TYPE.NULL);  }
    {reservedWord}  { return symbol(TOKEN_TYPE.RESERVED_WORD);}
    {typeName}      { return symbol(TOKEN_TYPE.TYPE);}
    {char}          { return symbol(TOKEN_TYPE.CHAR);}
    {identifier}    { return symbol(TOKEN_TYPE.ID);   }
    {int}       { return symbol(TOKEN_TYPE.INT, Integer.parseInt(yytext()) );  }
    {float}         { return symbol(TOKEN_TYPE.FLOAT); }
    {lineComment}   {}
    "{-"            { yybegin(COMMENT);              }
    "=="            { return symbol(TOKEN_TYPE.EQUAL);}
    "="             { return symbol(TOKEN_TYPE.EQ);   }
    ";"             { return symbol(TOKEN_TYPE.SEMI); }
```

Figure 2: Expressões Regulares

# 4 Considerações Finais

Até o momento da entrega deste relatório somente foi implementado o Analisador Léxico. Com o decorrer da disciplina outras partes do compilador serão implementadas. A documentação dos outros artefatos serão acrescidsa neste mesmo relatório.