Universidade Federal de Juiz de Fora

Departamento de Ciência da Computação

Relatório Trabalho Prático

Teoria dos Compiladores

Relatório Trabalho Prático

Analisador Léxico

André Luiz dos Reis - 201965004C

Lucca Oliveira Schröder - 201765205C

Juiz de Fora - MG

2023

Sumário

1. Introdução

O desenvolvimento desse relatório é com o intuito de abordar sobre o avanço da primeira etapa do trabalho onde iremos criar um compilador para a linguagem lang. Nesta primeira nós implementamos um analisador léxico.

A análise léxica é um processo onde é analisado uma sequência de caracteres de entrada e eles são convertidos em uma sequência de tokens (elementos básicos da linguagem, como identificadores, números, palavras-chave, caracteres, entre outros).

Temos essa análise léxica como a primeira etapa no processo de compilação de um programa, tendo como objetivo facilitar a análise sintática, reduzindo a quantidade de informações que precisam ser processadas. E um programa que realiza essa análise lexical também pode ser chamado de tokenizer, scanner ou lexer.

2. Implementação do Analisador Léxico

Aqui iremos abordar de forma mais detalhada como fizemos o desenvolvimento do nosso analisador léxico.

Na implementação utilizamos a linguagem Java e fizemos uso da ferramenta JFlex para poder gerar o analisador léxico. Para realizar os testes utilizamos exemplos anteriormente disponibilizados onde temos um arquivo de texto contendo

um código e o nosso analisador implementado deve percorrer caractere a caractere e imprimir a sequência dos lexemas.

2.1 Classes desenvolvidas

- **Main:** Na classe Main é onde primeiramente nós validamos o arquivo que ta sendo passado e logo após criamos a classe do analisador léxico. O arquivo é processado e ocorre a busca por novos tokens utilizando a função ***nextToken()***.

- **TokenType:** É onde ocorre a especificação dos tokens da linguagem. É feito por meio de uma enumeração.

- **Token:** Classe onde são carregados os tokens. Nela nós guardamos as informações do token, como, linha, coluna, lexema, um elemento da enumeração de tokens feita na classe TokenType e alguma informação a mais que possa ser colocada.

- **ArquivoJFlexLang:** Nesse arquivo são definidos algumas especificações (que serão abordadas de forma mais clara no tópico seguinte). Nele também é gerado de forma automática a classe **LexicalAnalyserLang,** de acordo com as especificações que foram passadas.

3. JFlex

Temos que o JFlex é uma ferramenta onde é gerado o analisador léxico na linguagem Java.

Na nossa classe ArquivoJFlexLang.jflex nós fizemos as definições dos macros,

que são expressões regulares que terão um nome escolhido anteriormente. A imagem abaixo demonstra os macros que foram passados e como nós os definimos.

```
/* Definição de macros */
ID_GERAL      = [:letter:] | [:digit:] | "_"
FimDeLinha    = \r|\n|\r\n
Brancos       = {FimDeLinha} | [ \t\f]
numeroDecimal = [:digit:]* "." [:digit:]+
numeroInteiro = [:digit:] [:digit:]*
identificador = [:lowercase:] {ID_GERAL}*
tipo          = [:uppercase:] {ID_GERAL}*
ASPAS         = [\'] | ["'"] | [']
chacter       = {ASPAS} (.) {ASPAS} | {ASPAS} ("\\n" | "\\t" | "\\b" | "\\r" | "\\\\")  {ASPAS}
LineComment   = "--" (.)* {FimDeLinha}
```

Além disso nós definimos as relações das entradas associadas aos seus tokens, podendo ser utilizado algum macro definido anteriormente ou algum outro tipo de entrada. No caso abaixo temos algumas palavras reservadas que utilizamos e os seus respectivos retornos.

```
/*  PALAVRAS RESERVADAS */
    "true" { return symbol(TOKEN_TYPE.TRUE_KEYWORD); }
    "false" { return symbol(TOKEN_TYPE.FALSE_KEYWORD); }
    "null" { return symbol(TOKEN_TYPE.NULL_KEYWORD); }
    "Bool" { return symbol(TOKEN_TYPE.BOOLEAN_KEYWORD); }
    "if" { return symbol(TOKEN_TYPE.IF_KEYWORD); }
    "else" { return symbol(TOKEN_TYPE.ELSE_KEYWORD); }
    "Float" { return symbol(TOKEN_TYPE.FLOAT_KEYWORD); }
    "Int" { return symbol(TOKEN_TYPE.INT_KEYWORD); }
    "data" { return symbol(TOKEN_TYPE.DATA_KEYWORD); }
    "Char" { return symbol(TOKEN_TYPE.CHAR_KEYWORD); }
    "print" { return symbol(TOKEN_TYPE.PRINT_KEYWORD); }
    "return" { return symbol(TOKEN_TYPE.RETURN_KEYWORD); }
    "read" { return symbol(TOKEN_TYPE.READ_KEYWORD); }
    "iterate" { return symbol(TOKEN_TYPE.ITERATE_KEYWORD); }
```

No caso dos macros nós utiizamos a função yytext() que já é nativa do JFlex e essa função retorna um ponteiro para uma string contendo o último token identificado pelo analisador léxico, com ela nós obtemos o lexema que está sendo analisado. Nós fazemos essa definição da forma como é mostrado na imagem a seguir.

```
/* IDENTIFICADOS GERAL */

    {identificador} { return symbol(TOKEN_TYPE.ID, yytext());   }
    {tipo}          { return symbol(TOKEN_TYPE.TYPE, yytext());   }
    {numeroDecimal} { return symbol(TOKEN_TYPE.DEC, Float.parseFloat(yytext()) );  }
    {numeroInteiro} { return symbol(TOKEN_TYPE.NUM, Integer.parseInt(yytext()) );  }
    {chacter}       { return symbol(TOKEN_TYPE.CHAR, yytext());   }
    "{-*"            { yybegin(COMMENT);}
}
```

4. Compilação e execução

Para a execução basta utilizar o comando **./run.sh $1,** sendo $1 o caminho para o arquivo a ser lido, considerando como raiz a pasta que está o arquivo .sh.