UNIOESTE – Universidade Estadual do Oeste do Paraná

Curso de Ciência da Computação

**Projeto 1**

**Compiladores**

**LARISSA LEWARTOSKI WONG**

Foz do Iguaçu – PR

2022

**Nome do Software:**

* Analisador léxico para subconjunto de C: Lex

**Nome dos integrantes da equipe:**

* O projeto foi realizado de forma individual.

Larissa Lewartoski Wong.

**Descrição da linguagem:**

A linguagem X é formada por 33 classes de tokens, sendo elas:

* Um token de encerramento do arquivo;
* Tokens para os operadores:
  + \*
  + /
  + +
  + –
  + <
  + <=
  + >
  + >=
  + %
  + ==
  + !
  + !=
  + &&
  + ||
* Tokens para os símbolos:
* (
* )
* {
* }
* ;
* ,
* Tokens para palavras-chave, reservadas ou tipos de dados:
* if
* Else
* While
* Tipodado (engloba os tipos int, float, char e double)
* Token de identificador
* Token de número
* Token para string

Os nomes das classes de tokens são:

Finaldaentrada, Op\_igual, Op\_multiplicacao, Op\_divisao, Op\_mod, Op\_soma, Op\_subtracao, Op\_not, Op\_menor, Op\_menorigual, Op\_maior, Op\_maiorigual, Op\_igual, Op\_naoigual, Op\_atribuicao, Op\_and, Op\_or, Palavrachave\_if, Palavrachave\_else, Palavrachave\_while, Abparentese, Fcparentese, Abchave, Fcchave, Pontoevirgula, Virgula, Identificador, Inteiro, String, Tipodado, Palavrachave\_for, Palavrachave\_return e Palavrachave\_include.

As classes que começam com Op todas identificam operadores.

As classes que começam com Palavrachave ou Tipodado são de palavras-chave, palavras reservadas ou tipos de dado da linguagem, foram incluídas as palavras while, if, else, for, return, include e os tipos de dado int, float, double e char.

A classe do token de finalização do arquivo é utilizada quando ocorre a entrada de um caractere inválido ou o arquivo lido acaba.

São classes de símbolos as classes com nomes dos próprios símbolos como a classe Virgula. No caso de classes com símbolos em que há abertura e fechamento, como por exemplo o caso de parênteses, foi adotada a forma Ab para sinalizar token de abertura e Fc para sinalizar token de fechamento.

O token de identificador aceita identificadores que se iniciem com letras (a até z), números (0 a 9) ou o caractere \_. Seguidos de mais caracteres seguindo a mesma regra.

O token de número aceita números de 0 a 9.

O token de string aceita linhas com o formato de string da linguagem.

Descrição das expressões regulares ou autômatos utilizados:

Para a maior parte dos tokens não foi necessário gerar expressões regulares complexas. No caso de tokens como os operadores, os símbolos e as palavras especiais as expressões são idênticas às palavras.

Para tokens como identificador, número e string foi necessário gerar expressão regular.

Assim temos:

Finaldaentrada = ( \u0000 )

Op\_igual = ( = )

Op\_multiplicacao = ( \* )

Op\_divisao = ( / )

Op\_mod = ( % )

Op\_soma = ( + )

Op\_subtracao = ( - )

Op\_not = ( ! )

Op\_menor = ( < )

Op\_menorigual = ( <= )

Op\_maior = ( > )

Op\_maiorigual = ( >= )

Op\_igual = ( = )

Op\_naoigual = ( !=)

Op\_atribuicao = ( = )

Op\_and = ( && )

Op\_or = ( || )

Palavrachave\_if = ( if )

Palavrachave\_else = ( else )

Palavrachave\_while = ( while )

Abparentese = ( ( )

Fcparentese = ( ) )

Abchave = ( { )

Fcchave = ( } )

Pontoevirgula = ( ; )

Virgula = ( , )

Identificador = [a – z | A – Z | \_ ] ( a – z | A – Z | \_ )\*

Inteiro = [0 – 9]+

String = “[ ^” \n]\*”

Tipodado = ( int | float | double | char )

Palavrachave\_for = ( for )

Palavrachave\_return = ( return )

Palavrachave\_include. = ( include )

Partes do código em que são definidos os tokens:

Tela de computador

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Tela de computador

Descrição gerada automaticamente

Tela de computador

Descrição gerada automaticamente

**Descrição do funcionamento do software:**

O software realiza a análise léxica a partir de um arquivo texto. Após a leitura do arquivo as informações são processadas e o software imprime no terminal as informações de cada token encontrado. Se tudo correr sem erros o arquivo é lido completamente, se for encontrado algum erro esse erro é impresso no terminal juntamente com o local do código em que ele se encontra. Todos os erros que aparecerem são encontrados.

**Instruções para a execução:**

Para rodar o código basta ter o compilador para a linguagem Java, pois o trabalho foi desenvolvido nessa linguagem. É necessário compilar. Se for de interesse compilar utilizando comando, para realizar esse processo é necessário acessar o local do arquivo lex.java e executar o seguinte comando:

**javac lex.java**

Esse comando realiza a compilação e gera arquivos .class, para que seja utilizado o caminho correto do arquivo no comando sugere-se que digite javac lex e utilize a tecla tab para completar o caminho do arquivo. Após a compilação é possível executar o programa com o comando:

**java Lex**

Com a execução o programa vai imprimir uma mensagem pedindo que seja digitado o nome ou caminho do arquivo em questão. Não é necessário digitar o final (.txt), apenas o nome ou caminho do arquivo desejado. Exemplos de inserção de arquivo:

**C:\Users\nomeUsuario\Desktop\teste1**

**teste1**

**Tratamento de erros:**

A estratégia de tratamento de erros é: identificar o erro, mostrar na tela a posição do erro (linha e coluna). Os erros identificados pelo programa são:

* Identificação de caractere vazio
* Sequência de escape não reconhecida (\r)
* Caractere não reconhecido
* Constante multi caracteres
* Final de arquivo enquanto lia string (fechamento de string faltando ou fora do final da linha)
* Número inválido (começando com número porém continuando com outros caracteres como 785abaco)

**Processo de construção:**

O processo de construção do software foi iniciado utilizando a ferramenta auxiliar JavaCC, porém ao final do código gerado foi localizada uma falha na leitura de números inválidos, na qual o programa acabava por separar a palavra. Exemplo: 123bacon aparecia como número e identificador, ao invés de ocorrer o erro. Por conta da dificuldade em encontrar o problema e após maiores pesquisas foi encontrado um exemplo com implementação e desenvolvido a partir deste ponto.

Após a realização de testes com o código encontrado como exemplo online foi decidido realizar adequações para que funcionasse com a linguagem pensada e do modo definido na especificação do trabalho.

As bibliotecas java.io.File e java.io.FileNotFoundException são utilizadas para manipulação do arquivo txt e geração de excessão caso o arquivo não seja encontrado.

As bibliotecas java.util.HashMap e java.util.Map são utilizadas na criação de um HashMap com as strings e tipos de tokens, para que os tokens fiquem guardados na lista. E Map faz parte da ligação das chaves aos valores.

Por fim a biblioteca java.util.Scanner é utilizada para a entrada do nome/caminho do arquivo e da leitura das informações de dentro do arquivo texto.

Tal como enfatizado anteriormente para utilizar o software é necessário ter o compilador para linguagem Java no computador. Também é necessário ter o Java instalado para que a linguagem seja interpretada posteriormente.

No caso da utilização do JavaCC é necessária a instalação e alteração em variáveis de ambiente do computador, porém essa solução foi descartada. Não foi utilizada IDE para o desenvolvimento, apenas o editor de texto Visual Studio Code e todos os testes e compilação foram realizados por meio de terminal (Powershell).

**Referências:**

**HashMap Java: Trabalhando com Listas key-value.** Disponível em: <https://www.devmedia.com.br/hashmap-java-trabalhando-com-listas-key-value/29811 >

**HashMap (Java Platform SE 8** **)**. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html>

**Java.io.File Class in Java - GeeksforGeeks**. Disponível em: <https://www.geeksforgeeks.org/file-class-in-java/>

**java - Making a lexical Analyzer - Stack Overflow**. Disponível em: <https://stackoverflow.com/questions/17848207/making-a-lexical-analyzer>

**Implement simple lexical analysis in Java (linuxtut.com**). Disponível em: <https://linuxtut.com/en/d7ea5d0e0dbc0409a01f/>

**C program to detect tokens in a C program - GeeksforGeeks.** Disponível em: https://www.geeksforgeeks.org/c-program-detect-tokens-c-program/

**JavaCC.** Disponível em: <https://javacc.github.io/javacc/>

**p01.analise-lexica.pdf (ufop.br).** Disponível em: <http://www.decom.ufop.br/romildo/2014-1/bcc328/slides/p01.analise-lexica.pdf>

**Experimento 1: Diseño e implementación del programa de análisis léxico (implementación de Java) - programador clic.** Disponível em: <https://programmerclick.com/article/60381479536/>

**Compiler/lexical analyzer - Rosetta Code**. Disponível em: <https://rosettacode.org/wiki/Compiler/lexical\_analyzer>