**Nome do software:** Analisador Léxico

**Integrantes do grupo:** Lucas Garavaglia

**Descrição da linguagem:** C é uma linguagem de programação compilada de propósito geral, estruturada, imperativa, procedural, padronizada pela Organização Internacional para Padronização (ISO), criada em 1972 por Dennis Ritchie na empresa AT&T Bell Labs para desenvolvimento do sistema operacional Unix (originalmente escrito em Assembly).

**Descrição do Autômato:** Foi criado um autómato universal que lê e executa uma sequência de estados previamente declarados. Foram criadas 12 classes de estados referente a cada classe de token.

**Funcionamento do software:** O software funciona com base em autômatos, assim que iniciar o programa com o comando npm start dentro do diretório principal irá aparecer uma mensagem com orientações, siga a orientação colocando o caminho com o nome do arquivo exemplo: /home/compiladores/trabalho/src/fonte1.txt, após inserido o caminho ele era ler e fazer a análise léxica, se houver algum erro ele irá indicar, caso contrário ele faz a análise sintática, se houver algum erro ele indicará na tela,se não houver

**Erros:** Após uma breve pesquisa foi definido os seguintes tipos de erros léxicos: palavra/token não pertence a nenhuma classe de tokens válidos, identificador mal formado e número mal formado.

**Build:** Para o funcionamento do código não será preciso compilar, somente usar o comando “npm install” para instalar bibliotecas e “npm start” para iniciar o programa, dentro do diretório principal do projeto, para o desenvolvimento do trabalho foi utilizado o editor de texto VScode na versão 1.5.4 e a linguagem javascript com a framework nodejs na versão 16.04 e biblioteca readline-sync na versão 1.4.10.

**Classe de tokens:**

Token para Início de bloco: [{]

Token para Final de bloco: [}]

Token para Início de Função: [(]

Token para final de Função: [)]

Token para separação: [,]

Token para Loop: [while]

Token para Condição: [if]

Token para Final de função: [return]

Token para Tipo de dado: [int]

Token para Atribuição: [=]

Token para Expressões Lógicas: [>=|<=|==|!=|>|<]

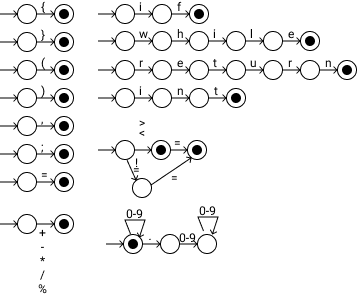
Token para Operadores matemáticos: [+|-|\*|/|%]

Token para Final de linha: [;]

Token para Variável: [\_|a-z|A-Z][\_|a-z|A-Z|0-9]\*

Token para Números: [0-9]+[[.][0-9]+]?

Essas classes de tokens foram definidas no arquivo “tokenClass.js” em forma de autômato finito.

**Regras Sintáticas:**

<STA> ::= tokenConditional tokenStartFunction<EXP>tokenFinalFunction<STA>

| tokenStartBlockFunction<STA>tokenFinalBlockFunction<STA>

| tokenWhile tokenStartFunction<EXP>tokenFinalFunction<STA>

| tokenReturn<EXP>tokenEndLine<STA>

| tokenEndLine<STA>

| tokenIdentifier<K>tokenEndLine<STA>

| tokenDataType tokenIdentifier<T>tokenEndLine<STA>

| tokenUnsigned tokenDataType tokenIdentifier<T>tokenEndLine<STA>

| tokenTypeDef tokenDataType tokenIdentifier tokenEndLine<STA>

| $

<T> ::= tokenAssignments<EXP>

| tokenSeparator tokenIdentifier<T>

<EXP> ::= tokenIdentifier<S>

| tokenStartFunction <EXP> tokenFinalFunction

| tokenNumber<S>

<K> ::= tokenAssignments<EXP>

| tokenStartFunction tokenIdentifier<B> tokenFinalFunction

<B> ::= tokenSeparator tokenIdentifier<B>

| $

<S> ::= tokenOperator<EXP>

| tokenExpression<EXP>

| $

<F> ::= tokenDataType tokenIdentifier tokenStartFunction<P>tokenFinalFunction<STA>

| tokenUnsigned tokenDataType tokenIdentifier tokenStartFunction<P>tokenFinalFunction<STA>

<P> ::= tokenDataType tokenIdentifier<Z>

| $

<Z> ::= tokenSeparator tokenIdentifier<Z>

| $

<TD> ::= tokenTypeDef tokenDataType tokenIdentifier tokenEndLine

Essas regras foram definidas no arquivo parser, dentro do construtor da classe parser, em forma de um objeto, onde cada objeto e um estado, e dentro de cada estado tem um vetor com as regras respectivas, por exemplo:

"<B>": [

["tokenSeparator", "tokenIdentifier", "<B>"],

["$"],

]

**Exemplo de uma árvore sintática gerado pelo analisador:**

**b = 3+4;**

**Lista de símbolos : $ tokenEndLine tokenNumber tokenOperator tokenNumber tokenAssignments tokenIdentifier**

**Pilha : <STA>**

**Lista de símbolos : $ tokenEndLine tokenNumber tokenOperator tokenNumber tokenAssignments tokenIdentifier**

**Pilha : <STA> tokenEndLine <K> tokenIdentifier**

**Lista de símbolos : $ tokenEndLine tokenNumber tokenOperator tokenNumber tokenAssignments**

**Pilha : <STA> tokenEndLine <K>**

**Lista de símbolos : $ tokenEndLine tokenNumber tokenOperator tokenNumber tokenAssignments**

**Pilha : <STA> tokenEndLine <EXP> tokenAssignments**

**Lista de símbolos : $ tokenEndLine tokenNumber tokenOperator tokenNumber**

**Pilha : <STA> tokenEndLine <EXP>**

**Lista de símbolos : $ tokenEndLine tokenNumber tokenOperator tokenNumber**

**Pilha : <STA> tokenEndLine <S> tokenNumber**

**Lista de símbolos : $ tokenEndLine tokenNumber tokenOperator**

**Pilha : <STA> tokenEndLine <S>**

**Lista de símbolos : $ tokenEndLine tokenNumber tokenOperator**

**Pilha : <STA> tokenEndLine <EXP> tokenOperator**

**Lista de símbolos : $ tokenEndLine tokenNumber**

**Pilha : <STA> tokenEndLine <EXP>**

**Lista de símbolos : $ tokenEndLine tokenNumber**

**Pilha : <STA> tokenEndLine <S> tokenNumber**

**Lista de símbolos : $ tokenEndLine**

**Pilha : <STA> tokenEndLine <S>**

**Lista de símbolos : $ tokenEndLine**

**Pilha : <STA> tokenEndLine**

**Lista de símbolos : $**

**Pilha : <STA>**

**Lista de símbolos : $**

**Pilha : $**

**Referências:**

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>

<https://www.npmjs.com/package/readline-sync>

Slides do professor.