**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,por fim é feito a análise semântica, caso tenha algum erro semântico e exibido na tela, caso contrário exibe uma mensagem indicando que a análise terminou.

**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, para os erros sintáticos é exibido na tela o token esperado e a linha onde esse token deveria estar, e para os erros semânticos é verificado variáveis declaradas e tipagem nas atribuições feitas.

**Gramática de atributos:** Na gramática de atributo foi calculado os tipos de dados de todos os tokens referentes, se o token não tem um tipo específico o valor é undefined.

**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, no sistema operacional Linux.

**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: [>=|<=|==|!=|>|<]

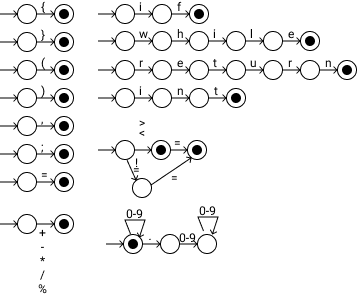
Token para Operadores matemáticos: [+|-|\*|/|%]

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<EXPFUNC>

| tokenStartBlockFunction<STA><ENDBLOCK>

| tokenWhile tokenStartFunction<EXPFUNC>

| tokenReturn<EXP>tokenEndLine<STA>

| tokenIdentifier<K>tokenEndLine<STA>

| tokenDataType <DECFUNC>

| tokenUnsigned tokenDataType <DECFUNC>

| tokenTypeDef <TD> <STA>

| $

<ENDBLOCK> ::= <STA>tokenFinalBlockFunction

<EXPFUNC> ::= <EXP>tokenFinalFunction tokenStartBlockFunction<STA><ENDBLOCK> {Definição do escopo}

<T> ::= tokenAssignments<EXP>

| tokenSeparator tokenIdentifier<T>

<L> ::= tokenStartFunction<P>tokenFinalFunction

|<T>tokenEndLine

<DECFUNC> ::= tokenIdentifier <L> <STA>

<EXP> ::= tokenIdentifier<S>

| tokenStartFunction<EXP>tokenFinalFunction

| tokenNumber<S>

<K> ::= tokenAssignments<EXP>

| tokenStartFunction tokenIdentifier<B>tokenFinalFunction

<B> ::= tokenSeparator tokenIdentifier<B>

| $

<S> ::= tokenOperator<EXP>

| tokenExpression<EXP>

| $

<P> ::= tokenDataType tokenIdentifier<Z>

| $

<Z> ::= tokenSeparator tokenDataType tokenIdentifier<Z>

| $

<TD> ::= 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 : tokenIdentifier tokenAssignments tokenNumber tokenOperator tokenNumber tokenEndLine $**

**Pilha : <STA>**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**Lista de símbolos : tokenNumber tokenEndLine $**

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

**Lista de símbolos : tokenNumber tokenEndLine $**

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

**Lista de símbolos : tokenEndLine $**

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

**Lista de símbolos : tokenEndLine $**

**Pilha : tokenEndLine <STA>**

**Lista de símbolos : $**

**Pilha : <STA>**

**Lista de símbolos : $**

**Pilha : $**

**Regras da gramática de atributo:**

| <STA> ::= tokenConditional tokenStartFunction<EXPFUNC> | <STA> ::= tokenConditional tokenStartFunction<EXPFUNC> |
| --- | --- |
| <STA> ::= tokenStartBlockFunction<STA><ENDBLOCK> | <STA> ::= tokenStartBlockFunction<STA><ENDBLOCK> |
| <STA> ::= tokenWhile tokenStartFunction<EXPFUNC> | <STA> ::= tokenWhile tokenStartFunction<EXPFUNC> |
| <STA> ::= tokenReturn<EXP>tokenEndLine<STA> | <STA> ::= tokenReturn<EXP>tokenEndLine<STA> |
| <STA> ::= tokenIdentifier<K>tokenEndLine<STA> | <STA> ::= tokenIdentifier<K>tokenEndLine<STA> |
| <STA> ::= tokenDataType <DECFUNC> | <STA> ::= tokenDataType <DECFUNC> |
| <STA> ::= tokenUnsigned tokenDataType <DECFUNC> | <STA> ::= tokenUnsigned tokenDataType <DECFUNC> |
| <STA> ::= tokenTypeDef <TD> <STA> | <STA> ::= tokenTypeDef <TD> <STA> |
| <ENDBLOCK> ::= <STA>tokenFinalBlockFunction | <ENDBLOCK> ::= <STA>tokenFinalBlockFunction |
| <EXPFUNC> ::= <EXP>tokenFinalFunction tokenStartBlockFunction<STA><ENDBLOCK> | <EXPFUNC> ::= <EXP>tokenFinalFunction tokenStartBlockFunction<STA><ENDBLOCK> |
| <T> ::= tokenAssignments<EXP> | <T> ::= tokenAssignments<EXP> |
| <T> ::= tokenSeparator tokenIdentifier<T> | <T> ::= tokenSeparator tokenIdentifier<T> |
| <L> ::= tokenStartFunction<P>tokenFinalFunction | <L> ::= tokenStartFunction<P>tokenFinalFunction |
| <L> ::= <T>tokenEndLine | <L> ::= <T>tokenEndLine |
| <DECFUNC> ::= tokenIdentifier <L> <STA> | tokenIdentifier.tipoData = <L>.tipoData <STA> |
| <EXP> ::= tokenIdentifier<S> | <EXP>.tipoData = tokenIdentifier.tipoData <S>.tipoData |
| <EXP> ::= tokenStartFunction<EXP>tokenFinalFunction | <EXP> ::= tokenStartFunction<EXP>tokenFinalFunction |
| <EXP> ::= tokenNumber<S> | <EXP>.tipoData = tokenNumber.tipoData <S> |
| <K> ::= tokenAssignments<EXP> | = <EXP>.tipoData |
| <K> ::= tokenStartFunction tokenIdentifier<B>tokenFinalFunction | <K> ::= tokenStartFunction tokenIdentifier<B>tokenFinalFunction |
| <B> ::= tokenSeparator tokenIdentifier<B> | <B> ::= tokenSeparator tokenIdentifier<B> |
| <S> ::= tokenOperator<EXP> | <S> ::= tokenOperator<EXP> |
| <S> ::= tokenExpression<EXP> | <S> ::= tokenExpression<EXP> |
| <P> ::= tokenDataType tokenIdentifier<Z> | tokenIdentifier.tipoData = tokenDataType |
| <Z> ::= tokenSeparator tokenDataType tokenIdentifier<Z> | tokenIdentifier.tipoData = tokenDataType <Z>.tipoData |
| <TD> ::= tokenDataType tokenIdentifier tokenEndLine | tokenIdentifier.tipoData = tokenDataType |

**Referências:**

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

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

Slides do professor.