﻿README.doc - Descrição do trabalho

**Nome do software:** Compilador

**Integrantes da equipe:** 1.Levi Cicero Arcanjo

**Descrição da linguagem X:**

• Subconjunto de C, incluindo declaração de variáveis int, char, float. Permitindo o uso dos operadores de soma, multiplicação, subtração, divisão, operadores lógicos maior e menor, comandos condicionais (if e else) e loops (while e for).

• Nomes variáveis podendo iniciar com qualquer letra de a-z, A-Z ou underline seguidas por qualquer letra de a-z, A-Z, underline e números.

• Palavras reservadas são: int, char, float, void, if, else, while, main, return e for

**Classes de Tokens:**

Palavra int -> Expressão regular: ^[i][n][t]$

Palavra char -> Expressão regular: ^[c][h][a][r]$

Palavra float -> Expressão regular: ^[f][l][o][a][t]$

Palavra void -> Expressão regular: ^[v][o][i][d]$

Palavra if -> Expressão regular: ^[i][f]$

Palavra else -> Expressão regular: ^[e][l][s][e]$

Palavra while -> Expressão regular: ^[w][h][i][l][e]$

Palavra return -> Expressão regular: ^[r][e][t][u][r][n]$

Palavra for -> Expressão regular: ^[f][o][r]$

símbolo abre parêntesis -> Expressão regular: ^[(]$

Símbolo fecha parênteses -> Expressão regular: ^[)]$

Símbolo abre chaves -> Expressão regular: ^[{]$

Símbolo fecha chaves -> Expressão regular: ^[}]$

Símbolo de soma -> Expressão regular: ^[+]$

Símbolo de subtração -> Expressão regular: ^[-]$

Símbolo de multiplicação -> Expressão regular: ^[\*]$

Símbolo de divisão -> Expressão regular: ^[/]$

Símbolo ponto e vírgula -> Expressão regular: ^[;]$

Símbolo de vírgula -> Expressão regular: ^[,]$

Símbolo de atribuição -> Expressão regular: ^[=]$

Símbolo and -> Expressão regular: ^[&]$

Símbolo ou -> Expressão regular: ^[|]$

Símbolo maior -> Expressão regular: ^[>]$

Símbolo menor -> Expressão regular: ^[<]$

Símbolo abre colchetes -> Expressão regular: ^[[]$

Símbolo fecha colchetes -> Expressão regular: ^[\]]$

Caracter -> Expressão regular: ^['].[']$

String -> Expressão regular: ^["].\*["]$

Número -> Expressão regular: ^[0-9]+$

Número decimal -> Expressão regular: ^[0-9]+[.][0-9]+$

Identificador -> Expressão regular: ^[\_a-z][\_a-z0-9]\*$

**Trecho de código associados às classes de tokens:**

A variável “tokenClasses” é atribuída com expressões regulares que serão utilizadas no programa. Neste código a variável “tokenClasses” é um array imutável de objetos, sendo cada objeto composto por uma nomenclatura da classe de token (atributo “tokenClass”) e sua expressão regular correspondente (atributo “re”).



**Regras sintáticas em notação similar a EBNF:**

Notação não-terminais: <NÃO-TERMINAIS>

Notação terminais: terminais

Notação palavra vazia: λ

| Regras Sintáticas | | |
| --- | --- | --- |
| <S> | ::= | <TYPE> <IDENTIFIER> <S0> <S> |
| <S> | ::= | λ |
| <S0> | ::= | ; <S0\_> |
| <S0> | ::= | <FUNCTION\_> |
| <DECLARATION> | ::= | <TYPE> <IDENTIFIER> <DECLARATION\_> |
| <DECLARATION\_> | ::= | ; |
| <DECLARATION\_> | ::= | , <IDENTIFIER> <DECLARATION\_> |
| <IDENTIFIER> | ::= | id |
| <TYPE> | ::= | int |
| <TYPE> | ::= | float |
| <TYPE> | ::= | char |
| <TYPE> | ::= | void |
| <VALUE> | ::= | number |
| <VALUE> | ::= | decimal |
| <VALUE> | ::= | string |
| <VALUE> | ::= | character |
| <FUNCTION> | ::= | <TYPE> <IDENTIFIER> <FUNCTION\_> |
| <FUNCTION\_> | ::= | ( <F0> ) <STATEMENT> |
| <F0> | ::= | <TYPE> <IDENTIFIER> <F1> |
| <F0> | ::= | λ |
| <F1> | ::= | , <F0> |
| <F1> | ::= | [ <F2> ] |
| <F1> | ::= | λ |
| <F2> | ::= | number |
| <F2> | ::= | λ |
| <ITERATION> | ::= | while ( <EXPRESION> ) <STATETMENT> |
| <ITERATION> | ::= | for ( <ITERATION\_> ; <ITERATION\_> ; <ITERATION\_> ) <STATEMENT> |
| <SELECTION> | ::= | if ( <EXPRESION> ) <STATEMENT> <ELSE> |
| <ELSE> | ::= | else <STATEMENT> |
| <ELSE> | ::= | λ |
| <OPERATOR> | ::= | | + | - | \* | / | < | > | = | & |
| <STATEMENT\_\_> | ::= | <SELECTION> |
| <STATEMENT\_\_> | ::= | <ITERATION> |
| <STATEMENT\_\_> | ::= | <EXPRESION> ; |
| <STATEMENT\_\_> | ::= | <RETURN> |
| <RETURN> | ::= | return <ITERATION\_> ; |
| <S0\_> | ::= | <S> |
| <S0\_> | ::= | λ |
| <ASSIGNMENT> | ::= | <PRIMARY> <ASSIGNMENT\_> |
| <ASSIGNMENT\_> | ::= | <OPERATOR> <ASSIGNMENT> |
| <ASSIGNMENT\_> | ::= | λ |
| <PRIMARY> | ::= | <IDENTIFIER> |
| <PRIMARY> | ::= | <VALUE> |
| <EXPRESION> | ::= | <ASSIGNMENT> <EXPRESION\_> |
| <EXPRESION\_> | ::= | , <ASSIGNMENT> <EXPRESION\_> |
| <EXPRESION\_> | ::= | λ |
| <STATEMENT> | ::= | { <STATEMENT\_> } |
| <STATEMENT\_> | ::= | <STATEMENT\_\_> <STATEMENT\_> |
| <STATEMENT\_> | ::= | <DECLARATION> <STATEMENT\_> |
| <STATEMENT\_> | ::= | λ |
| <ITERATION\_> | ::= | <EXPRESION> |
| <ITERATION\_> | ::= | λ |

**Trecho de código associados à regra gramatical:**

Regra associada neste trecho de código:

<EXPRESION\_> :: = , <ASSIGNMENT> <EXPRESION\_> | λ



**Regras sintáticas e gramática de atributos correspondente:**

| Regras Sintáticas | | | Gramática de atributos |
| --- | --- | --- | --- |
| <S> | ::= | <TYPE> <IDENTIFIER> <S0> <S> | <IDENTIFIER>.type = <TYPE> |
| <S> | ::= | λ |  |
| <S0> | ::= | ; <S0\_> |  |
| <S0> | ::= | <FUNCTION\_> |  |
| <DECLARATION> | ::= | <TYPE> <IDENTIFIER> <DECLARATION\_> | <IDENTIFIER>.type = <TYPE> |
| <DECLARATION\_> | ::= | ; |  |
| <DECLARATION\_> | ::= | , <IDENTIFIER> <DECLARATION\_> |  |
| <IDENTIFIER> | ::= | id | id.type = <IDENTIFIER>.type |
| <TYPE> | ::= | int |  |
| <TYPE> | ::= | float |  |
| <TYPE> | ::= | char |  |
| <TYPE> | ::= | void |  |
| <VALUE> | ::= | number | <TYPE>.type = int |
| <VALUE> | ::= | decimal | <TYPE>.type = float |
| <VALUE> | ::= | string | <TYPE>.type = char |
| <VALUE> | ::= | character | <TYPE>.type = char |
| <FUNCTION> | ::= | <TYPE> <IDENTIFIER> <FUNCTION\_> | <IDENTIFIER>.type = <TYPE> |
| <FUNCTION\_> | ::= | ( <F0> ) <STATEMENT> |  |
| <F0> | ::= | <TYPE> <IDENTIFIER> <F1> | <IDENTIFIER>.type = <TYPE> |
| <F0> | ::= | λ |  |
| <F1> | ::= | , <F0> |  |
| <F1> | ::= | [ <F2> ] |  |
| <F1> | ::= | λ |  |
| <F2> | ::= | number |  |
| <F2> | ::= | λ |  |
| <ITERATION> | ::= | while ( <EXPRESION> ) <STATETMENT> | <ITERATION>.scope = currentScope + 1 |
| <ITERATION> | ::= | for ( <ITERATION\_> ; <ITERATION\_> ; <ITERATION\_> ) <STATEMENT> | <ITERATION>.scope = currentScope + 1 |
| <SELECTION> | ::= | if ( <EXPRESION> ) <STATEMENT> <ELSE> | <SELECTION>.scope = currentScope + 1 |
| <ELSE> | ::= | else <STATEMENT> | <ELSE>.scope = currentScope + 1 |
| <ELSE> | ::= | λ |  |
| <OPERATOR> | ::= | | + | - | \* | / | < | > | = | & |  |
| <STATEMENT\_\_> | ::= | <SELECTION> |  |
| <STATEMENT\_\_> | ::= | <ITERATION> |  |
| <STATEMENT\_\_> | ::= | <EXPRESION> ; |  |
| <STATEMENT\_\_> | ::= | <RETURN> |  |
| <RETURN> | ::= | return <ITERATION\_> ; |  |
| <S0\_> | ::= | <S> |  |
| <S0\_> | ::= | λ |  |
| <ASSIGNMENT> | ::= | <PRIMARY> <ASSIGNMENT\_> | <PRIMARY>.type = <ASSIGNMENT\_>.type, <ASSIGNMENT>.value = <PRIMARY>.value |
| <ASSIGNMENT\_> | ::= | <OPERATOR> <ASSIGNMENT> |  |
| <ASSIGNMENT\_> | ::= | λ |  |
| <PRIMARY> | ::= | <IDENTIFIER> | <PRIMARY>.type = <IDENTIFIER>.type |
| <PRIMARY> | ::= | <VALUE> | <PRIMARY>.type = <VALUE>.type |
| <EXPRESION> | ::= | <ASSIGNMENT> <EXPRESION\_> | <EXPRESION>.value = <ASSIGNMENT>.value |
| <EXPRESION\_> | ::= | , <ASSIGNMENT> <EXPRESION\_> |  |
| <EXPRESION\_> | ::= | λ |  |
| <STATEMENT> | ::= | { <STATEMENT\_> } |  |
| <STATEMENT\_> | ::= | <STATEMENT\_\_> <STATEMENT\_> |  |
| <STATEMENT\_> | ::= | <DECLARATION> <STATEMENT\_> |  |
| <STATEMENT\_> | ::= | λ |  |
| <ITERATION\_> | ::= | <EXPRESION> |  |
| <ITERATION\_> | ::= | λ |  |

**Descrição do funcionamento e utilização do software:**

- Funcionamento: funciona em um navegador (preferência chrome ou firefox)

- Execução: Abrir o index.html com o navegador (preferência chrome ou firefox)

- Uso: selecionar o arquivo clicando no botão "selecionar arquivo", clicar no botão "Fazer análise" para iniciar a análise (léxica, sintática e semântica). Caso existam erros, serão mostrados na tela. Caso não existam erros de compilação será iniciado o download do arquivo da árvore de derivação sintática.

**Descrição do tratamento de erros:**

**Erros léxicos:**

Neste analisador os erros léxicos são identificados através de expressões regulares, logo após ser constatado que o token está mal escrito é chamado um método denominado describeLexicalError() que determina qual tipo de erro o token apresenta.

Este analisador léxico reconhece os seguintes erros especificamente:

- Identificador mal formado -> Quando um identificador inicia com números

- Número mal formado-> Quando existe um número com letras e pontos

- Caráter mal formado -> Quando um caractere possui tamanho maior que um ou aspas simples a menos

- String mal formada -> Quando possui aspas duplas a menos

- Símbolo desconhecido -> Quando o símbolo apresentado não faz parte da linguagem

- Tamanho excessivo -> Quando o símbolo apresentado ultrapassa a quantidade permitida de caracteres.

**Erros sintáticos:**

Neste analisador sintático os erros são identificados através de métodos recursivos. Cada método possui comandos de seleção que verificam a ordem esperada de tokens durante a análise. Cada método recursivo, quando detecta a ocorrência de um erro, determina de forma específica o tipo de erro com base no token esperado. Caso o token seja diferente do esperado é empilhado um erro e a análise ignora esse token faltante, seguindo a análise sintática como se não houvessem ocorrido erros até o momento.

Os erros sintáticos são descritos por uma mensagem que diz qual o token esperado seguido de sua linha e o token anterior.

**Erros Semânticos:**

Neste analisador semântico os erros são identificados de forma específica, ou seja, para cada tipo de erro existe um trecho que código único para detectá-lo e tratá-lo.

Os erros semânticos detectados neste analisador são:

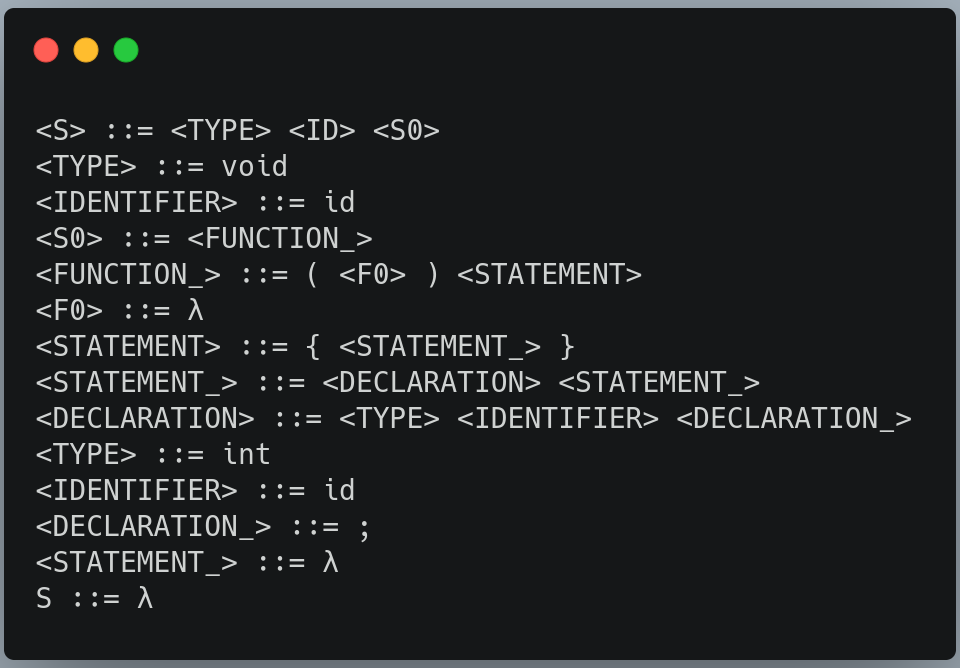
* Identificador já declarado
* Identificador não declarado
* Divergência de tipos

**Descrição do processo de construção:**

- Não é necessário nenhum tipo de processo de build

- Apenas é necessário ter instalado um navegador web

**Árvore sintática(imagem a esquerda) referente um trecho de código (imagem a direita):**





**Referências usadas:**

- https://www.w3schools.com/

- https://developer.mozilla.org/

- https://javascript.info/

- https://pt.stackoverflow.com/

- Slides de compiladores (Aulas 0, 1 e 2, 3, 4, 5, 6)