Analisador Léxico

Luiz Henrique Cardoso - 11621853
Maurílio Barboza Martins - 11621220

Análise léxica é o processo de analisar a entrada de linhas de caracteres e produzir uma sequência de símbolos chamado "símbolos léxicos", ou somente "símbolos", que podem ser manipulados mais facilmente por um parser. Esse símbolos podem também ser chamados de Tokens.

Nosso analisador Léxico é composto pelas classes, Identificador,Ts, Token e Lexer, além de um estrutura de enumeração.

Classe Token: é a classe responsável por retornar os símbolos léxicos identificados pelo analisador, cada token é composto de um TAG <"Tipo do símbolo", "Símbolo reconhecido">.

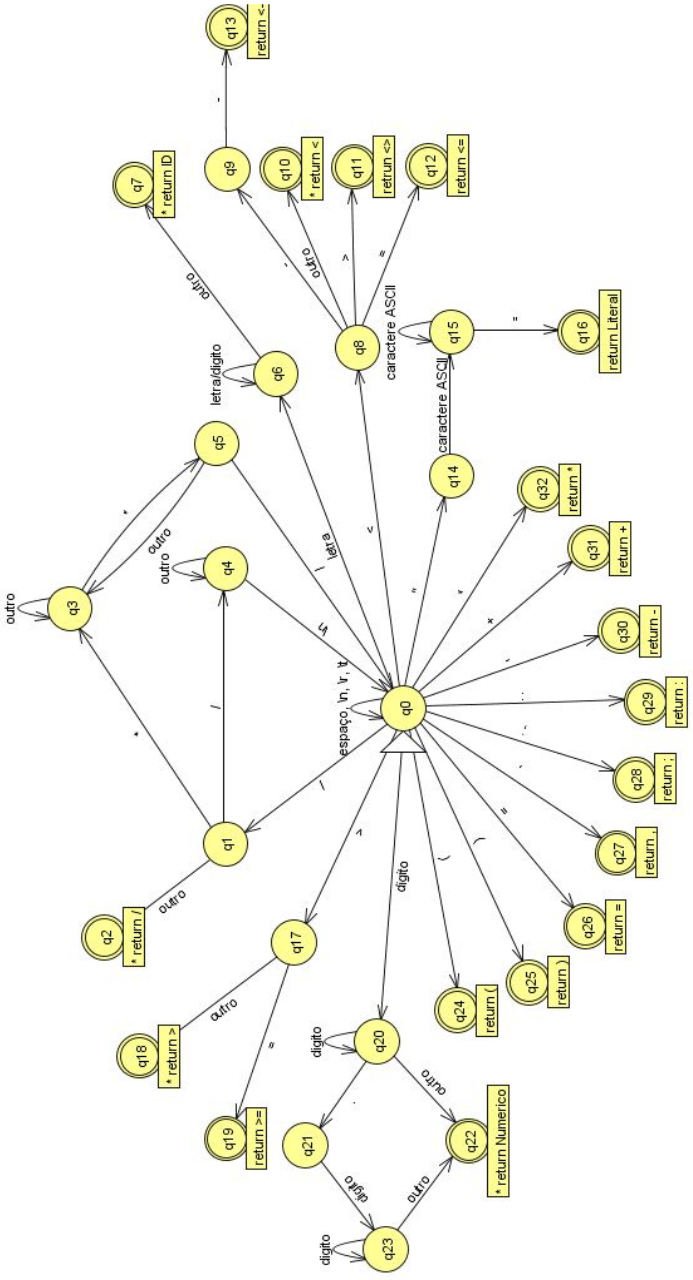
Classe TS: a classe Ts é a classe onde são cadastradas as palavras reservadas da linguagem, ela possui uma estrutura de HashMap, que armazenar os tokens de todas as palavras reservadas. Também possui um método que cadastrar novos tokens e um método que retorna os tokens cadastrados.

Classe Lexer: é a principal classe do analisador, nessa classe é realizada a leitura do arquivo(caractere por caractere) e todas as verificações para retornando os tokens para os símbolos identificados e também retornando os erros encontrados no código. Ela possui duas funções que manipulam a abertura e o fechamento dos arquivos. Uma função que retorna erro para o usuário e uma que torna o ponteiro indicando qual linha e coluna a leitura no caractere tem que ser feita. Além disso existe a função que faz a leitura dos caracteres, ela faz a leitura dos caracteres do código fornecido pela usuario contro a posição onde está ocorrendo essa leitura, e identifica se o caracter é final ou se é um símbolo formado por mais caracteres, dessa forma ao ler uma caractere ela pode imediatamente retornar um token ou ler o próximo caractere para formar um símbolo ou esperado reconhecido pela gramática ela. Ela repete esse processo até encontrar um erro ou fim de arquivo.

Classe Identificador: é a classe mais simples do analisador sua função consiste em armazenar o identificador de cada token.

Estrutura de enumeração: é utilizada para armazenar os tipos de tokens que podem ser reconhecidos pela linguagem.

AFD:


q0
espaço, \n, \r, \t
letra
digito
q1
q2
* return /
outro
q3
outro
q4
outro
q5
letra/digito
q6
q7
* return ID
outro
q8
caractere ASCII
q9
-
q10
* return <
q11
return <>
q12
return <=
q13
return <-
q14
caractere ASCII
q15
=
q16
return Literal
q17
>
=
q18
* return >
outro
q19
return >=
q20
digito
outro
q21
digito
outro
q22
* return Numerico
q23
digito
outro
q24
return (
q25
return)
q26
return =
q27
return ,
q28
return ;
q29
return :
q30
return -
q31
return +
q32
return *

Gramática:

Compilador → Programa EOF

Programa → "algoritmo" ("declare" (DeclaraVar)+)? ListaCmd "fim" "algoritmo" ListaRotina

DeclaraVar → ListaID Tipo ";"

ListaRotina → ListaRotina'

ListaRotina' → Rotina ListaRotina' | ε

Rotina → "subrotina" ID "(" ListaParam ")" ("declare" (DeclaraVar)+)? ListaCmd Retorno "fim" "subrotina"

ListaParam → Param "," ListaParam | Param

Param → ListaID Tipo

ListaID → ID "," ListaID | ID

Retorno → "retorne" Expressao | ε

Tipo → "logico" | "numerico" | "literal" | "nulo"

ListaCmd → ListaCmd Cmd

ListaCmd' → Cmd ListaCmd' | ε

Cmd → CmdSe | CmdEnquanto | CmdPara | CmdRepita | CmdAtrib | CmdChamaRotina | CmdEscreva | CmdLeia

CmdSe →"se" "(" Expressao ")" "inicio" ListaCmd "fim" | "se" "(" Expressao ")" "inicio" ListaCmd "fim" "senao" "inicio" ListaCmd "fim"

CmdEnquanto → "enquanto" "(" Expressao ")" "faca" "inicio" ListaCmd "fim"

ComandoPara → "para" CmdAtrib "ate" Expressao "faca" "inicio" ListaCmd "fim"

ComandoRepita → "repita" ListaCmd "ate" Expressao

CmdAtrib → ID "<--" Expressao ";"

CmdChamaRotina → ID "(" (Expressao ("," Expressao)*)? ")" ";"

CmdEscreva → "escreva" "(" Expressao ")" ";"

CmdLeia → "leia" "(" ID ")" ";"

Expressao → Expressao Op Expressao | ID | ID "(" (Expressao ("," Expressao)*)? ")" | Numerico | Literal | "verdadeiro" | "falso" | OpUnario Expressao | "(" Expressao ")"

Op → "Ou" | "E" | "<" | "<=" | ">" | ">=" | "=" | "<>" | "/" | "*" | "-" | "+"

OpUnario → "Nao"