**UNIVERSIDADE FEDERAL DE UBERLÂNDIA**

**Trabalho de Construção de Compiladores**

**Projeto da Linguagem**

**Aluno:** Marcelo Mendonça Borges

**Matrícula:** 11611BCC020

**2019**

**Gramática da Linguagem**

Foi utilizada a estrutura definida no projeto para se construir a gramática. A gramática é definida pela 4-upla abaixo:

G = (V,T,S,P)

1. V 🡪 {inicio, bloco, declaracao, comando, tipo, id, tamanho, int, char, real, expressao\_parentesis, expressao, posicao, condicao\_parentesis, condicao, condição\_para, arg}
2. T 🡪 {PROGRAMA, INICIO, FIM, “,”, “;”, “(”, “)”, “[“, “]”, SE, ENQUANTO, FACA, PARA, ==, <>, <, >, >=, <=, NOT, AND, OR, +, -, \*, /, INT, REAL, CHAR, STRING, WS}
3. S 🡪 inicio
4. A definição de P, considerando a notação EBNF:

inicio

: 'PROGRAMA' bloco

bloco

: 'INICIO' declaracao comando 'FIM'

;

declaracao

: tipo id ';'

| tipo '[' tamanho ']' id ';'

| tipo (id ',')\* id ';'

| tipo '[' tamanho ']' (id ',')\* id ';'

;

tipo

: int

| char

| real

;

tamanho

: int

;

comando

: id '=' expressao\_parentesis ';'

| id '[' posicao ']' '=' expressao\_parentesis ';'

| 'SE' condicao\_parentesis bloco

| 'ENQUANTO' condicao\_parentesis bloco

| 'FACA' bloco 'ENQUANTO' condicao\_parentesis ';'

| 'PARA' condicao\_para bloco

;

condicao\_parentesis

: '(' condicao ')'

;

condicao

: arg

| arg '==' arg

| arg '<>' arg

| arg '<' arg

| arg '>' arg

| arg '>=' arg

| arg '<=' arg

;

expressao\_parentesis

: '(' expressao ')'

;

expressao

: arg

| arg '+' expressao\_parentesis

| arg '-' expressao\_parentesis

| arg '\*' expressao\_parentesis

| arg '/' expressao\_parentesis

;

posicao

: int

;

condicao\_para

: '(' id '=' expressao ';' condicao ';' id '=' expressao ')'

;

arg

: id

| int

| real

| condicao\_parentesis

| expressao\_parentesis

;

id

: [a-zA-Z]+

;

int

: INT

;

char

: CHAR

;

real

: REAL;

;

INT

: [0-9]+

| '-' [0-9]+

;

CHAR

: [a-zA-Z]

;

REAL

: [0-9]+ '.' [0-9]+

| '-' [0-9]+ '.' [0-9]+

;

WS

: [ \n\t] -> skip

;

**Tokens**

1. Palavras Reservadas: PROGRAMA, INICIO, FIM, SE, ENQUANTO, FACA, PARA, NOT, AND, OR, INT, CHAR, REAL.

|  |  |  |
| --- | --- | --- |
| **Lexemas** | **Token** | **Valor do Atributo** |
| PROGRAMA | PROGRAMA | - |
| INICIO | INICIO | - |
| FIM | FIM | - |
| , | virgula | - |
| ; | ponto\_virgula | - |
| ( | abre\_parentesis | - |
| ) | fecha\_parentesis | - |
| [ | abre\_colchete | - |
| ] | fecha\_colchete | - |
| SE | SE | - |
| ENQUANTO | ENQUANTO | - |
| FACA | FACA | - |
| PARA | PARA | - |
| == | igual | - |
| <> | diferente | - |
| < | menor | - |
| > | maior | - |
| >= | maior\_igual | - |
| <= | menor\_igual | - |
| + | soma | - |
| - | subtracao | - |
| \* | multiplicacao | - |
| / | divisao | - |
| INT | INT | valor inteiro dentro dos limites |
| REAL | REAL | valor real dentro dos limites |
| CHAR | CHAR | qualquer letra |
| (Qualquer) WS | - | - |
| (Qualquer) id | id | Posição na tabela de símbolos |

**Padrões dos Tokens**

Estarei considerando as seguintes definições:

1. numero 🡪 [0-9]
2. letra 🡪 [a-zA-Z]

As expressões regulares que representam os padrões mais importantes são:

1. id 🡪 STRING
2. INT 🡪 (-)? numero+
3. REAL 🡪 (-)? numero+ .numero+
4. CHAR 🡪 letra
5. WS 🡪 [ \t\n]\*

As expressões regulares dos demais tokens são essencialmente a concatenação de cada caractere do token, pois eles irão aparecer no código exatamente desta forma.