UNIVERSIDADE FEDERAL DO PAMPA - UNIPAMPA
Curso Bacharelado em Engenharia de Computação
Disciplina de Linguagens Formais

Professor(a): Ana Paula Ludtke Ferreira

RELATÓRIO

Analisador léxico e sintático em java

Arthur Teixeira Jardim
Marcelo Marchioro Cordeiro
Rodrigo Acosta Segui
Matheus Soares

Bagé, 08 de julho de 2019.

Resumo:

Esse relatório apresenta o detalhamento das atividades que foram realizadas ao decorrer da implementação de um analisador léxico e sintático para disciplina de Linguagens Formais do curso de Engenharia de Computação na Universidade Federal do Pampa no Campus Bagé. Esse projeto tem como objetivo utilizar os conhecimentos adquiridos na disciplina e aplicá-los na construção de um analisador léxico e sintático de uma linguagem *L* (inspirada na sintaxe da linguagem LISP) gerada pela gramática livre de contexto que foi fornecida pela docente da disciplina em questão. A partir da leitura de um arquivo texto que conterá uma linha de código da linguagem ocorre a validação dos analisadores, onde o léxico deve criar uma lista de tokens e o sintático verificar se o programa escrito pertence à linguagem.

Analisador Léxico e Sintático:

O trabalho consistiu na implementação de um analisador léxico e sintático a partir da linguagem linguagem *L* (Inspirada na sintaxe da linguagem LISP) gerada pela seguinte gramática de contexto:

Tabela 1 - Sintaxe da linguagem *L*

<programa>	::=	<lista-de-funções:>
<lista-de-funções>	::=	<função>
<lista-de-funções>	::=	<função><lista-de-funções>
<função>	::=	(defun <id> (<params>) <corpo>)
<corpo>	::=	(if <cond> <corpo> <corpo>)
<corpo>	::=	(<id> <corpo>)
<corpo>	::=	<exp>
<params>	::=	<id> <params>
<params>	::=	<id>
<exp>	::=	<num>
<exp>	::=	<id>
<exp>	::=	(+ <exp> <exp>)
<exp>	::=	(- <exp> <exp>)
<exp>	::=	(* <exp> <exp>)
<exp>	::=	(/ <exp> <exp>)
<cond>	::=	(gt <exp><exp>)

<cond>	::=	(lt <exp><exp>)
<cond>	::=	(geq <exp><exp>)
<cond>	::=	(leq <exp><exp>)
<cond>	::=	(eq <exp><exp>)
<cond>	::=	(neq <exp><exp>)
<cond>	::=	(not <exp><exp>)
<cond>	::=	(and <exp><exp>)
<cond>	::=	(or <exp><exp>)
<id>	::=	<letra><seqsimb>
<num>	::=	<digito>
<num>	::=	<digito><num>
<letra>	::=	a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<digito>	::=	0|1|2|3|4|5|6|7|8|9
<seqsimb>	::=	ε
<seqsimb>	::=	<letra><seqsimb>
<seqsimb>	::=	<digito> <seqsimb>

1º Passo: Foi realizada a leitura do arquivo que possui um programa escrito na linguagem *L*, o arquivo é lido uma linha por vez que ao final é concatenada em uma variável do tipo String denominada *expr*. Essa variável é utilizada para realizar a análise léxica no programa.

Figura 1 - Leitura do arquivo,

```
String expr = "";
try{
    BufferedReader br = new BufferedReader(new FileReader("arquivo.txt"));
    while(br.ready()){
        String linha = br.readLine();
        expr += linha;
    }
    br.close();
}catch(IOException ioe){
    ioe.printStackTrace();
}
```

Fonte: autores (2019).

2º Passo: Para o desenvolvimento do analisador léxico, foi necessário criar uma classe do tipo enum contendo os tokens utilizados são eles: *DelimEsq, DelimDir, OpUn, OpBin, TesteCond, NomeFunc, Num, ID, Espaço, palavrareservada*.

Figura 2 - Tokens.

```
public enum Tokens {
    DelimEsq, DelimDir, OpUn, OpBin, TesteCond,
    NomeFunc, Num, ID, Espaco,palavrareservada,
}
```

Fonte: autores (2019).

O analisador léxico foi desenvolvido com base na biblioteca JFlex, onde criamos um arquivo chamado language.flex, no arquivo foram colocados os símbolos terminais referentes a linguagem *L*. Na compilação do programa a biblioteca é capaz de de gerar a classe *LexicalAnalyzer.java* a partir do arquivo *language.flex*, a classe java é utilizada para a classificação dos tokens. Para analisar o código inserido no arquivo texto, é utilizado pela classe *LexicalAnalyzer* um *StringReader* que lê a String *expr* mencionada no 1º Passo, assim retornando os elementos do tipo enum em um *switch* que os classifica e imprime na tela.

3º Passo : Substituímos manualmente todas as regras que possuíam recursão à esquerda ou que poderiam ser fatoradas à esquerda por regras equivalentes.

Tabela 2 - Regras substituídas.

<programa>	::=	<lista-de-funções:>
<lista-de-funções>	::=	<função><lf>
<lf>	::=	<função><lf>
<lf>	::=	ε
<funções>	::=	(defun <id> (<params>) <corpo>)
<corpo>	::=	(if <cond> <corpo> <corpo>)
<corpo>	::=	(<id> <corpo>)
<corpo>	::=	<exp>
<params>	::=	<params><p>
<p>	::=	<id><p>
<p>	::=	ε
<exp>	::=	<num>
<exp>	::=	<id>

<exp>	::=	(+ <exp> <exp>)
<exp>	::=	(- <exp> <exp>)
<exp>	::=	(* <exp> <exp>)
<exp>	::=	(/ <exp> <exp>)
<cond>	::=	(gt <exp><exp>)
<cond>	::=	(lt <exp><exp>)
<cond>	::=	(geq <exp><exp>)
<cond>	::=	(leq <exp><exp>)
<cond>	::=	(eq <exp><exp>)
<cond>	::=	(neq <exp><exp>)
<cond>	::=	(not <exp><exp>)
<cond>	::=	(and <exp><exp>)
<cond>	::=	(or <exp><exp>)
<num>	::=	<digito><n>
<n>	::=	<digito><n>
<n>	::=	ε
<id>	::=	<letra><n>
<n>	::=	<digito><n>
<n>	::=	<letra><n>
<n>	::=	ε

4º Passo: Primeiramente criamos manualmente todas as regras de transição

1. (p, ε, ε, q, <programa>)

2.1 (q, (, ε, q(, ε)
2.2 (q,), ε, q), ε)
2.3 (q, defun, ε, qdefun, ε)
2.4 (q, if, ε, qif, ε)
2.5 (q, +, ε, q+, ε)

2.6 (q, -, ε, q-, ε)
2.7 (q, *, ε, q*, ε)
2.8 (q, /, ε, q/, ε)
2.9 (q, gt , ε, qgt, ε)
2.10 (q, geq, ε, qgeq, ε)
2.11 (q, leq, ε, qleq, ε)
2.12 (q, eq, ε, qeq, ε)
2.13 (q, and, ε, qand, ε)
2.14 (q, or, ε, qor, ε)
2.15 (q, neq, ε, qneq, ε)
2.16 (q, not, ε, qnot, ε)
2.17 (q, 0, ε, q0, ε)
2.18 (q, 1, ε, q1, ε)
2.19 (q, 2, ε, q2, ε)
2.20 (q, 3, ε, q3, ε)
2.21 (q, 4, ε, q4, ε)
2.22 (q, 5, ε, q5, ε)
2.23 (q, 6, ε, q6, ε)
2.24 (q, 7, ε, q7, ε)
2.25 (q, 8, ε, q8, ε)
2.26 (q, 9, ε, q9, ε)
2.27 (q, a, ε, qa, ε)
2.28 (q, b, ε, qb, ε)
2.29 (q, c, ε, qc, ε)
2.30 (q, d, ε, qd, ε)
2.31 (q, e, ε, qe, ε)
2.32 (q, f, ε, qf, ε)
2.33 (q, g, ε, qg, ε)
2.34 (q, h, ε, qh, ε)
2.35 (q, i, ε, qi, ε)
2.36 (q, j, ε, qj, ε)
2.37 (q, k, ε, qk, ε)
2.38 (q, l, ε, ql, ε)
2.39 (q, m, ε, qm, ε)
2.40 (q, n, ε, qn, ε)
2.41 (q, o, ε, qo, ε)
2.42 (q, p, ε, qp, ε)
2.43 (q, q, ε, qq, ε)
2.44 (q, r, ε, qr, ε)
2.45 (q, s, ε, qs, ε)
2.46 (q, t, ε, qt, ε)
2.47 (q, u, ε, qu, ε)
2.48 (q, v, ε, qv, ε)
2.49 (q, w, ε, qw, ε)
2.50 (q, x, ε, qx, ε)
2.51 (q, y, ε, qy, ε)

2.52 (q, z, ε, qz, ε)
2.53 (q, “ ”, ε, qEsp, ε)

3.1 (q(, ε, (, q, ε)
3.2 (q), ε,), q, ε)
3.3 (qdefun, ε, defun, q, ε)
3.4 (qif, ε, if, q, ε)
3.5 (q+, ε, +, q, ε)
3.6 (q-, ε, -, q, ε)
3.7 (q*, ε, *, q, ε)
3.8 (q/, ε, /, q, ε)
3.9 (qgt, ε, gt, q, ε)
3.10 (qgeq, ε, geq, q, ε)
3.11 (qleq, ε, leq, q, ε)
3.12 (qeq, ε, eq, q, ε)
3.13 (qand, ε, and, q, ε)
3.14 (qor, ε, or, q, ε)
3.15 (qneg, ε, neg, q, ε)
3.16 (qnot, ε, not, q, ε)
3.17 (q0, ε, 0, q, ε)
3.18 (q1, ε, 1, q, ε)
3.19 (q2, ε, 2, q, ε)
3.20 (q3, ε, 3, q, ε)
3.21 (q4, ε, 4, q, ε)
3.22 (q5, ε, 5, q, ε)
3.23 (q6, ε, 6, q, ε)
3.24 (q7, ε, 7, q, ε)
3.25 (q8, ε, 8, q, ε)
3.26 (q9, ε, 9, q, ε)
3.27 (qa, ε, a, q, ε)
3.28 (qb, ε, b, q, ε)
3.29 (qc, ε, c, q, ε)
3.30 (qd, ε, d, q, ε)
3.31 (qe, ε, e, q, ε)
3.32 (qf, ε, f, q, ε)
3.33 (qg, ε, g, q, ε)
3.34 (qh, ε, h, q, ε)
3.35 (qi, ε, i, q, ε)
3.36 (qj, ε, j, q, ε)
3.37 (qk, ε, k, q, ε)
3.38 (ql, ε, l, q, ε)
3.39 (qm, ε, m, q, ε)
3.40 (qn, ε, n, q, ε)
3.41 (qo, ε, o, q, ε)
3.42 (qp, ε, p, q, ε)
3.43 (qq, ε, q, q, ε)

3.44 (qr, ε, r, q, ε)
3.45 (qs, ε, s, q, ε)
3.46 (qt, ε, t, q, ε)
3.47 (qu, ε, u, q, ε)
3.48 (qv, ε, v, q, ε)
3.49 (qw, ε, w, q, ε)
3.50 (qx, ε, x, q, ε)
3.51 (qy, ε, y, q, ε)
3.52 (qz, ε, z, q, ε)
3.53 (qEsp, ε, “ “, q, ε)

4.1 (qσ, ε,<programa>, qσ, <lista-de-funcoes>)
4.2 (qσ, ε, <lista-de-funcoes>, qσ, <funcao><lf>)
4.3 (qσ, ε, <lf>, qσ, <funcao><lf>)
4.4 (qσ, ε, <lf>, qσ, ε)
4.5 (q(, ε, <funcao>, q(,(defun <id> (<params>) <corpo>))
4.6 (q(, ε, <corpo>, q(, (if <cond> <corpo> <corpo>))
4.7 (q(, ε, <corpo>, q(, (<id> <corpo>))
4.8 (qσ, ε, <corpo>, qσ, <exp>)
4.9 (qσ, ε, <params>, qσ, <p>)
4.10 (qσ, ε, <p>, qσ, <id><p>)
4.11 (qσ, ε, <p>, qσ, ε)
4.12 (qσ, ε, <exp>, qσ, <num>)
4.13 (qσ, ε, <exp>, qσ, <id>)
4.14 (q(, ε, <exp>, q(, (+ <exp> <exp>))
4.15 (q(, ε, <exp>, q(, (- <exp> <exp>))
4.16 (q(, ε, <exp>, q(, (* <exp> <exp>))
4.17 (q(, ε, <exp>, q(, (/ <exp> <exp>))
4.18 (q(, ε, <cond>, q(, (gt <exp> <exp>))
4.19 (q(, ε, <cond>, q(, (lt <exp> <exp>))
4.20 (q(, ε, <cond>, q(, (geq <exp> <exp>))
4.21 (q(, ε, <cond>, q(, (leq <exp> <exp>))
4.22 (q(, ε, <cond>, q(, (eq <exp> <exp>))
4.23 (q(, ε, <cond>, q(, (neq <exp> <exp>))
4.24 (q(, ε, <cond>, q(, (not <exp> <exp>))
4.25 (q(, ε, <cond>, q(, (and <exp> <exp>))
4.26 (q(, ε, <cond>, q(, (or <exp> <exp>))
4.27 (qσ, ε, <num>, qσ, <digito><nn>)
4.28 (qσ, ε, <nn>, qσ, <digito><nn>)
4.29 (qσ, ε, <nn>, qσ, ε)
4.30 (qσ, ε, <id>, qσ, <letra><n>)
4.31 (qσ, ε, <n>, qσ, <digito><n>)
4.32 (qσ, ε, <n>, qσ, <letra><n>)
4.33 (qσ, ε, <n>, qσ, ε)
4.34 (q0, ε, <digito>, q0, 0)
4.35 (q1, ε, <digito>, q1, 1)

4.36 (q2, ε, <digito>, q2, 2)
4.37 (q3, ε, <digito>, q3, 3)
4.38 (q4, ε, <digito>, q4, 4)
4.39 (q5, ε, <digito>, q5, 5)
4.40 (q6, ε, <digito>, q6, 6)
4.41 (q7, ε, <digito>, q7, 7)
4.42 (q8, ε, <digito>, q8, 8)
4.43 (q9, ε, <digito>, q9, 9)
4.44 (qa, ε, <letra>, qa, a)
4.45 (qb, ε, <letra>, qb, b)
4.46 (qc, ε, <letra>, qc, c)
4.47 (qd, ε, <letra>, qd, d)
4.48 (qe, ε, <letra>, qe, e)
4.49 (qf, ε, <letra>, qf, f)
4.50 (qg, ε, <letra>, qg, g)
4.51 (qh, ε, <letra>, qh, h)
4.52 (qi, ε, <letra>, qi, i)
4.53 (qj, ε, <letra>, qj, j)
4.54 (qk, ε, <letra>, qk, k)
4.55 (ql, ε, <letra>, ql, l)
4.56 (qm, ε, <letra>, qm, m)
4.57 (qn, ε, <letra>, qn, n)
4.58 (qo, ε,<letra>, qo, o)
4.59 (qp, ε, <letra>, qp, p)
4.60 (qq, ε,<letra>, qq, q)
4.61 (qr, ε, <letra>, qr, r)
4.62 (qs, ε, <letra>, qa, s)
4.63 (qt, ε, <letra>, qt, t)
4.64 (qu, ε, <letra>, qu, u)
4.65 (qv, ε,<letra>, qv, v)
4.66 (qw, ε, <letra>, qw, w)
4.67 (qx, ε, <letra>, qx, x)
4.68 (qy, ε, <letra>, qy, y)
4.69 (qz, ε, <letra>, qz, z)

Após isso, as regras foram passadas ao programa em forma de matriz, uma para cada regra. Cada linha da matriz refere-se a uma regra e cada coluna refere-se a um termo da regra. Também se faz uso de um vetor de palavras reservadas para poder diferenciar um caracter qualquer de uma das palavras reservadas. O uso de duas pilhas deve-se ao fato de que na primeira inserção na pilha os dados são inserido na ordem contrária da que seria considerada a certa, logo se for desempilhando da pilha auxiliar e empilhando novamente em outra pilha tudo estará na ordem certa.

Figura 3 - Instanciando matrizes e vetor de palavras reservada,

```
String[][] reg1= new String[1][5];
String[][] reg2 = new String[52][5];
String[][] reg3 = new String[52][5];
String[][] reg4 = new String[69][5];
String[] palavraReservada = new String[50];
Setmatriz matriz = new Setmatriz();
```

Fonte: autores (2019).

A variável denominada controle tem a função de manter o programa rodando até chegar no estado final, então a mesma assume o valor de de true durante a execução do programa e ao chegar no estado final a mesma torna-se *false* e com isso se dá o encerramento do programa. Além dos estados terminais convencionais, criamos também o estado terminal *qEsp* para tratar os espaços no programa.

Como resultado é mostrado o passo a passo das transições que o programa faz seguindo as regras definidas para ele. Dependendo do estado, ele analisa o que tem na pilha ou o que recebe na entrada e vai para um estado seguinte respeitando a regra que se adequa para a situação. Foi utilizado o símbolo "#", para marcar o final do código, funcionando assim como o estado final.

Análise e Resultados dos programas testados:

Programa 1 : Entrada: (defun funcao (x) (+ 10 20))#

Figura 4 - Lista de tokens.

```
LISTA DE TOKENS

DelimEsq         | (
Palavra Terminal | defun
Espaço           |
ID               | fun
Espaço           |
DelimEsq         | (
ID               | xyz
DelimDir         | )
Espaço           |
DelimEsq         | (
Palavra Terminal | if
Espaço           |
DelimEsq         | (
TesteCond        | gt
Espaço           |
Num              | 2
Espaço           |
Num              | 3
DelimDir         | )
Espaço           |
ID               | a
Espaço           |
ID               | b
DelimDir         | )
DelimDir         | )
Simb final       | #
```

Fonte: autores (2019).

Figura 5 - Transições.

```
Estado: p | Entrada: (defun fun (xyz) (if (gt 2 3) a b))# | Pilha: vazia
Estado: q | Entrada: (defun fun (xyz) (if (gt 2 3) a b))# | Pilha: <programa>
Estado: q( | Entrada: defun fun (xyz) (if (gt 2 3) a b))# | Pilha: <programa>
Estado: q( | Entrada: defun fun (xyz) (if (gt 2 3) a b))# | Pilha: <lista-de-funcoes>
Estado: q( | Entrada: defun fun (xyz) (if (gt 2 3) a b))# | Pilha: <funcao>
Estado: q( | Entrada: defun fun (xyz) (if (gt 2 3) a b))# | Pilha: (
Estado: q | Entrada: defun fun (xyz) (if (gt 2 3) a b))# | Pilha: defun
Estado: qdefun | Entrada:  fun (xyz) (if (gt 2 3) a b))# | Pilha: defun
Estado: q | Entrada:  fun (xyz) (if (gt 2 3) a b))# | Pilha:
Estado: qEsp | Entrada: fun (xyz) (if (gt 2 3) a b))# | Pilha:
Estado: q | Entrada: fun (xyz) (if (gt 2 3) a b))# | Pilha: <id>
Estado: qf | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha: <id>
Estado: qf | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha: <letra>
Estado: qf | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha: f
Estado: q | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha: <n>
Estado: qu | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha: <n>
Estado: qu | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha: <letra>
Estado: qu | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha: u
Estado: q | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha: <n>
Estado: qn | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha: <n>
Estado: qn | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha: <letra>
Estado: qn | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha: n
Estado: q | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha: <n>
Estado: q | Entrada:  (xyz) (if (gt 2 3) a b))# | Pilha:
Estado: qEsp | Entrada: (xyz) (if (gt 2 3) a b))# | Pilha:
Estado: q | Entrada: (xyz) (if (gt 2 3) a b))# | Pilha: (
Estado: q( | Entrada: xyz) (if (gt 2 3) a b))# | Pilha: (
Estado: q | Entrada: xyz) (if (gt 2 3) a b))# | Pilha: <params>
Estado: qx | Entrada: ) (if (gt 2 3) a b))# | Pilha: <params>
Estado: qx | Entrada: ) (if (gt 2 3) a b))# | Pilha: <p>
Estado: qx | Entrada: ) (if (gt 2 3) a b))# | Pilha: <id>
Estado: qx | Entrada: ) (if (gt 2 3) a b))# | Pilha: <letra>
Estado: qx | Entrada: ) (if (gt 2 3) a b))# | Pilha: x
Estado: q | Entrada: ) (if (gt 2 3) a b))# | Pilha: <n>
```

```
Estado: qy | Entrada: ) (if (gt 2 3) a b))# | Pilha: <n>
Estado: qy | Entrada: ) (if (gt 2 3) a b))# | Pilha: <letra>
Estado: qy | Entrada: ) (if (gt 2 3) a b))# | Pilha: y
Estado: q | Entrada: ) (if (gt 2 3) a b))# | Pilha: <n>
Estado: qz | Entrada: ) (if (gt 2 3) a b))# | Pilha: <n>
Estado: qz | Entrada: ) (if (gt 2 3) a b))# | Pilha: <letra>
Estado: qz | Entrada: ) (if (gt 2 3) a b))# | Pilha: z
Estado: q | Entrada: ) (if (gt 2 3) a b))# | Pilha: <n>
Estado: q | Entrada: ) (if (gt 2 3) a b))# | Pilha: <p>
Estado: q) | Entrada:  (if (gt 2 3) a b))# | Pilha: <p>
Estado: q | Entrada:  (if (gt 2 3) a b))# | Pilha:
Estado: qEsp | Entrada: (if (gt 2 3) a b))# | Pilha:
Estado: q | Entrada: (if (gt 2 3) a b))# | Pilha: <corpo>
Estado: q( | Entrada: if (gt 2 3) a b))# | Pilha: <corpo>
Estado: q( | Entrada: if (gt 2 3) a b))# | Pilha: (
Estado: q | Entrada: if (gt 2 3) a b))# | Pilha: if
Estado: qif | Entrada:  (gt 2 3) a b))# | Pilha: if
Estado: q | Entrada:  (gt 2 3) a b))# | Pilha:
Estado: qEsp | Entrada: (gt 2 3) a b))# | Pilha:
Estado: q | Entrada: (gt 2 3) a b))# | Pilha: <cond>
Estado: q( | Entrada: gt 2 3) a b))# | Pilha: <cond>
Estado: q( | Entrada: gt 2 3) a b))# | Pilha: (
Estado: q | Entrada: gt 2 3) a b))# | Pilha: gt
Estado: qgt | Entrada:  2 3) a b))# | Pilha: gt
Estado: q | Entrada:  2 3) a b))# | Pilha:
Estado: qEsp | Entrada: 2 3) a b))# | Pilha:
Estado: q | Entrada: 2 3) a b))# | Pilha: <exp>
Estado: q2 | Entrada:  3) a b))# | Pilha: <exp>
Estado: q2 | Entrada:  3) a b))# | Pilha: <num>
Estado: q2 | Entrada:  3) a b))# | Pilha: <digito>
Estado: q2 | Entrada:  3) a b))# | Pilha: 2
Estado: q | Entrada:  3) a b))# | Pilha: <nn>
Estado: q | Entrada:  3) a b))# | Pilha:
Estado: qEsp | Entrada: 3) a b))# | Pilha:
Estado: q | Entrada: 3) a b))# | Pilha: <exp>
```

```
Estado: q3 | Entrada: ) a b))# | Pilha: <exp>
Estado: q3 | Entrada: ) a b))# | Pilha: <num>
Estado: q3 | Entrada: ) a b))# | Pilha: <digito>
Estado: q3 | Entrada: ) a b))# | Pilha: 3
Estado: q | Entrada: ) a b))# | Pilha: <nn>
Estado: q | Entrada: ) a b))# | Pilha: )
Estado: q) | Entrada:  a b))# | Pilha: )
Estado: q | Entrada:  a b))# | Pilha:
Estado: qEsp | Entrada: a b))# | Pilha:
Estado: q | Entrada: a b))# | Pilha: <corpo>
Estado: qa | Entrada:  b))# | Pilha: <corpo>
Estado: qa | Entrada:  b))# | Pilha: <exp>
Estado: qa | Entrada:  b))# | Pilha: <num>
Estado: qa | Entrada:  b))# | Pilha: <digito>
Estado: qa | Entrada:  b))# | Pilha: a
Estado: q | Entrada:  b))# | Pilha: <nn>
Estado: q | Entrada:  b))# | Pilha:
Estado: qEsp | Entrada: b))# | Pilha:
Estado: q | Entrada: b))# | Pilha: <corpo>
Estado: qb | Entrada: ))# | Pilha: <corpo>
Estado: qb | Entrada: ))# | Pilha: <exp>
Estado: qb | Entrada: ))# | Pilha: <num>
Estado: qb | Entrada: ))# | Pilha: <digito>
Estado: qb | Entrada: ))# | Pilha: b
Estado: q | Entrada: ))# | Pilha: <nn>
Estado: q | Entrada: ))# | Pilha: )
Estado: q) | Entrada: )# | Pilha: )
Estado: q | Entrada: )# | Pilha: )
Estado: q) | Entrada: # | Pilha: )
Estado: q | Entrada: # | Pilha: )
Estado: q# | Entrada:  | Pilha: )
Estado: q# | Entrada:  | Pilha: #
```

Dificuldades encontradas durante o desenvolvimento:

No início do projeto foi preciso criar as regras para as transições, e pelo fato de ser um número grande de regras uma das dificuldades foi gerar as mesmas e criar algo para controlar elas dentro do programa em java.

Tratar os espaços em branco e saber onde o programa terminava foi algo que dificultou o desenvolvimento do código e foi resolvido criando um estado a mais para cada um deles é um símbolo para representar o final do programa.

Problemas na parte de lógica foi um dos principais dificuldades na hora de desenvolver o algoritmo, sendo de bastante dificuldade desenvolver funções para a manipulação dos dados para o programa executar corretamente a análise seguindo as regras criadas.

Equipe executora:

- Arthur Teixeira Jardim, arthurtjardim@gmail.com, matrícula: 1701570231.
- Marcelo Marchioro Cordeiro, marcelo_1411@hotmail.com, matrícula: 1701560005.

- Rodrigo Acosta Segui, rodrigo_segui@hotmail.com, matrícula: 1701560120.
- Matheus Soares, mateus123soares@hotmail.com, matrícula: 1701570259.