

CENTRO UNIVERSITÁRIO FEI

JOÃO PEDRO ROSA CEZARINO - R.A: 22.120.021-5

HUGO LINHARES OLIVEIRA - R.A: 22.120.046-2

VITOR MARTINS OLIVEIRA - R.A: 22.120.067-8

THALES LACERDA OLIVEIRA - R.A: 22.120.056-1

RELATÓRIO DE PROJETO: Compiladores

São Bernardo do Campo

2022

SUMÁRIO

1	INTRODUÇÃO	3
2	DEFINIÇÃO DOS TOKENS E GRAMÁTICA	4
3	INSTRUÇÕES DE EXECUÇÃO DO COMPILADOR	6
4	EXEMPLO DE ENTRADA/SAÍDA DO COMPILADOR	7

1 INTRODUÇÃO

O presente projeto tem por objetivo a criação de um Compilador utilizando técnicas aprendidas durante as aulas da disciplina CC6252 - Compiladores do Centro Universitário FEI. O compilador deve ser capaz de realizar a conversão de um programa desenvolvido na Linguagem definida pelo grupo para a Linguagem Java, C ou Python. A linguagem para conversão definida pelo grupo foi a linguagem Java.

A verificação da corretude do programa será realizada compilando o arquivo gerado pelo compilador desenvolvido. O compilador deverá receber como entrada um arquivo contendo um programa escrito na Linguagem definida pelo grupo e gerar uma forma equivalente em Java, que deverá ser compilada no compilador *javac*, executada na JVM e não deverá conter erros.

Importante: A gramática não pode conter recursividade à esquerda. Caso seja necessário, efetue a sua fatoração à esquerda.

A linguagem criada pelo grupo foi nomeada de **L.G.L - Lazy Recursive Language** e utiliza termos e expressões utilizadas no dia a dia pelos alunos para gerar código Java após sua compilação.

2 DEFINIÇÃO DOS TOKENS E GRAMÁTICA

```

1 grammar atribuicao;
2
3 comece: comando*;
4
5 comando: comando_declaracao | comando_atribuicao | comando_print |
6 comando_op_logico | comando_matematico | comando_input | comando_op_do |
7 comando_op_logico_caso_nao;
8
9 read: 'fazer_leitura';
10
11 //===== PRINT =====//
12 comando_print: print l_par (id | texto) r_par termine comando*;
13
14 aspas: '"';
15 print: 'joga_na_tela';
16 l_par: '(';
17 r_par: ')';
18 l_keys: '{';
19 r_keys: '}';
20 texto: '"' id+ '"';
21
22 //===== OP MATEMATICA =====//
23 comando_matematico: (num | num_decimal) operador_matematico
24 (num | num_decimal) termine comando*;
25
26 operador_matematico: '+' | '*' | '/' | '-';
27
28 //===== ATRIBUICAO =====//
29 comando_declaracao: tipo comando_atribuicao;
30
31 comando_atribuicao: id operador (id|num|num_decimal) termine;
32
33 //===== INPUT =====//
34 comando_input: input l_par (id | texto) r_par termine comando*;
35
36 input: 'escreva';
37

```

```

38 //===== DECISAO =====//
39 comando_op_logico: (repeticao | condicional) l_par
40 (id|num|num_decimal|aspas id aspas) operador
41 (id|num|num_decimal|aspas id aspas) r_par l_keys
42 (comando) r_keys comando*;
43
44 comando_op_do: repeticao l_keys comando r_keys repeticao l_par
45 (id|num|num_decimal|aspas id aspas) operador
46 (id|num|num_decimal|aspas id aspas) r_par comando*
47 termine+;
48
49 comando_op_logico_caso_nao: condicional l_keys (comando) r_keys comando*;
50
51 condicional: 'caso_sim' | 'caso_nao' | 'caso_nao_e';
52
53 repeticao: 'enquanto' | 'fa a';
54
55 //===== DECISAO =====//
56
57 tipo: 'numero_inteiro' | 'numero_quebrado' | 'texto' | 'caracter' | '
    booleano';
58
59 operador: '=' | '<' | '>' | '!=';
60
61 termine: ' ';
62
63 id: ID;
64 ID: [a-z]+ | [A-Z]+;
65
66 num: NUM;
67 NUM: [0-9]+;
68
69 num_decimal: NUM_DECIMAL;
70 NUM_DECIMAL: [0-9]+.[0-9]+;
71
72 Ws: [ \t\r\n]+ -> skip;

```

atribuicao.g4

3 INSTRUÇÕES DE EXECUÇÃO DO COMPILADOR

Para uma execução mais automatizada do compilador, foi criado o script **lrl.sh**. Este script é responsável por executar os comandos de compilação do arquivo *atribuicao.g4*, que contém a definição da gramática e executar o *javac* nos arquivos gerados, assim como executá-los com o *java* após a finalização do processo.

Como funcionalidades adicionais, o script **lrl.sh** realiza a indentação do código gerado pelo compilador, utilizando o pacote *Artistic Style - astyle* e gera a árvore de tokens, no formato Postscript, do código passado como input, através do ANTLR4 TestRig - Grun.

Caso o pacote *Artistic Style - astyle* não seja encontrado no sistema em que o script está rodando, ele será instalado automaticamente, para garantir a correta execução.

Para executar o script, primeiro é necessário conceder as devidas permissões:

```
1 chmod u+x ./lrl.sh
```

A partir daí, podemos executar o script:

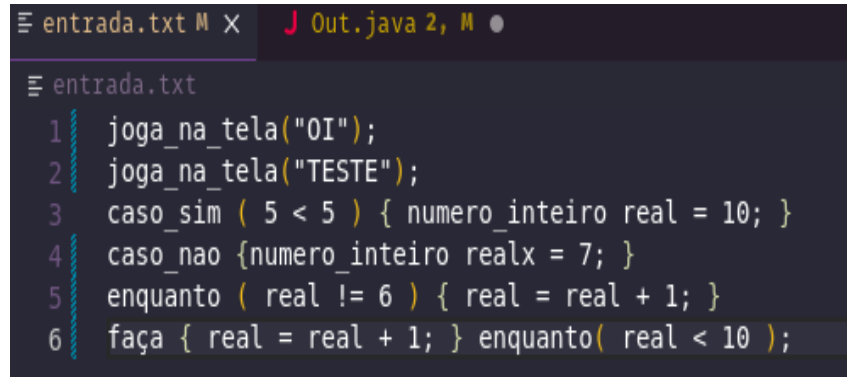
```
1 ./lrl.sh
```

OBS.: O arquivo utilizado como input para o script é o **entrada.txt**. Caso deseje utilizar outro arquivo como entrada, alterar as seguintes linhas no script **lrl.sh**:

```
1 java Main < entrada.txt  
2 java org.antlr.v4.gui.TestRig atribuicao comece -ps output_img.ps <  
   entrada.txt
```

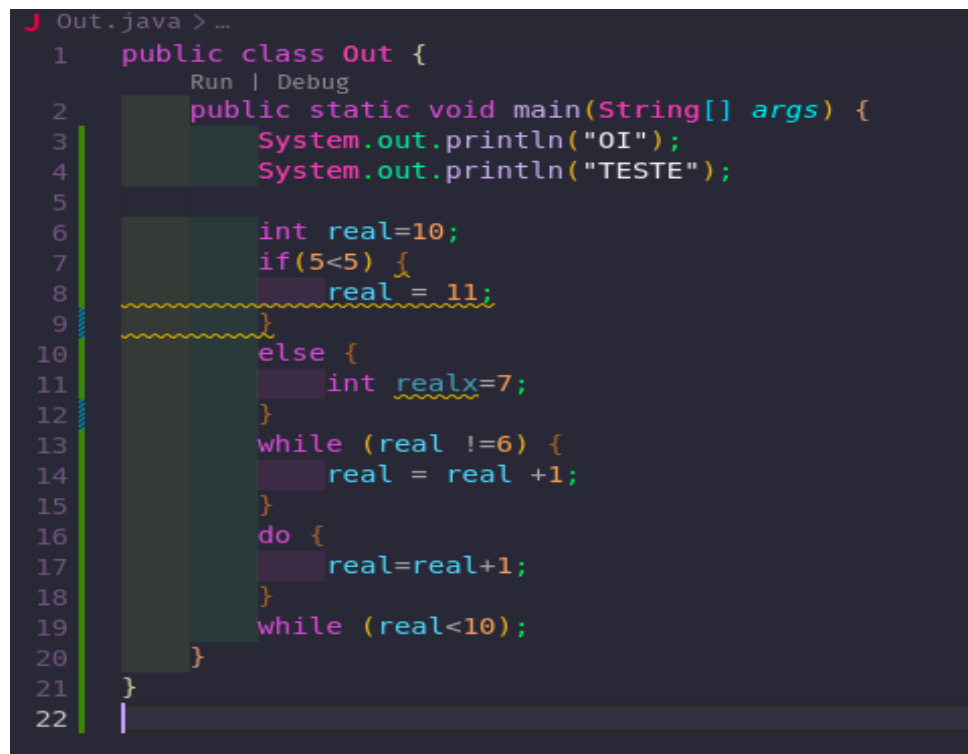
4 EXEMPLO DE ENTRADA/SAÍDA DO COMPILADOR

Entrada:



```
entrada.txt M X J Out.java 2, M ●
entrada.txt
1  joga_na_tela("OI");
2  joga_na_tela("TESTE");
3  caso_sim ( 5 < 5 ) { numero_inteiro real = 10; }
4  caso_nao {numero_inteiro realx = 7; }
5  enquanto ( real != 6 ) { real = real + 1; }
6  faça { real = real + 1; } enquanto( real < 10 );
```

Saída:



```
J Out.java > ...
1  public class Out {
2      Run | Debug
3      public static void main(String[] args) {
4          System.out.println("OI");
5          System.out.println("TESTE");
6
7          int real=10;
8          if(5<5) {
9              real = 11;
10         }
11         else {
12             int realx=7;
13         }
14         while (real !=6) {
15             real = real +1;
16         }
17         do {
18             real=real+1;
19         }
20         while (real<10);
21     }
22 }
```

