Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Compiladores I - DCC053
Trabalho Prático 1
Análise Léxica e Sintática

Marcel Henrique S. Mendes
(2013007641)

2º Semestre de 2018

Conteúdo

1 Descrição do Problema

Em projetos de compiladores modernos, as etapas iniciais do processo de análise do código fonte são as análises Léxica e Sintática. Durante a análise léxica, o programa fonte é separado em tokens, que são sequências significativas, que serão usadas posteriormente pela análise sintática da linguagem em questão. Com a identificação dos tokens, pode-se construir a tabela de símbolos necessária para a fase posterior, de análise semântica e de geração de código.

Já a fase de análise sintática é responsável por, a partir dos tokens produzidos inicialmente, impor uma estrutura gramatical sobre o programa fonte. Todo este processo é feito a partir da construção de árvores de derivação, descrita por uma gramática livre de contexto. Após esse processo é verificado se os tokens formam uma expressão permitida.

Nesse trabalho, são implementadas as fases de análise léxica e sintática para a linguagem de programação Simple, definida através da gramática mostrada na Figura 1.

```
program ::= program identifier ";" decl_list compound_stmt
decl_list ::= decl_list " ;" decl
            | decl
decl ::= dcl_var
       | dcl_proc
dcl_var ::= ident_list ":" type
ident_list ::= ident_list " ," identifier
             | identifier
type ::= integer
       | real
       | boolean
       | char
dcl_proc ::= tipo_retornado PROCEDURE identifier espec_parametros corpo
tipo_retornado ::= integer
                 | real
                 | boolean
                 | char
                 | vazio
corpo ::= ":" decl_list ";" compound_stmt id_return
        | vazio
id_return ::= identifier
            | vazio
espec_parametros ::= "(" lista_parametros ")"
lista_de_parametros ::= parametro
                      | lista_de_parametros , parametro
parametro ::= modo type : identifier
modo ::= value
       | reference
compound_stmt ::= begin stmt_list end
stmt_list ::= stmt_list ";" stmt
            | stmt
```

Figura 1: Gramática da linguagem para qual a análise léxica e sintática será feita.

2.

```
                |  compound_stmt
                |  function_ref_par
assign_stmt  ::=  identifier " :=" expr
if_stmt      ::=  if cond then stmt
                |  if cond then stmt
                   else stmt
cond         ::=  expr
repeat_stmt  ::=  repeat stmt_list until expr
read_stmt    ::=  read "(" ident_list ")"
write_stmt   ::=  write "(" expr_list ")"
expr_list    ::=  expr
                |  expr_list " ," expr
expr         ::=  Simple_expr
                |  Simple_expr RELOP Simple_expr
Simple_expr  ::=  term
                |  Simple_expr ADDOP term
term         ::=  factor_a
                |  term MULOP factor_a
factor_a     ::=  "−" factor
                |  factor
factor       ::=  identifier
                |  constant
                |  " (" expr " )"
                |  NOT factor
                |  function_ref_par
function_ref_par ::=  variable " (" expr_list " )"
variable     ::=  Simple_variable_or_proc
Simple_variable_or_proc ::=  identifier
constant     ::=  integer_constant
                |  real_constant
                |  char_constant
                |  boolean_constant
boolean_constant :=  false | true
```

Figura 2: Gramática da linguagem para qual a análise léxica e sintática será feita.

2 Metodologia

O trabalho foi realizado utilizando a linguagem de programação Java. Para esta, existem ferramentas que auxiliam no desenvolvimento de analisadores léxico e sintáticos. Nesse trabalho foram utilizadas as ferramentas JFLex e Cup, que funcionam de forma conjunta.

Para gerar analisadores léxicos, a ferramenta JFLex se mostra de grande utilidade, uma vez que permite que os tokens da linguagem sejam especificados através das expressões regulares correspondentes. Dessa forma, o problema de gerar um analisador léxico é reduzido ao problema de determinar expressões regulares para cada um dos tokens da linguagem.

Já para gerar analisadores sintáticos, a ferramenta CUP funciona de forma coordenada com o JFLex. Nela, basta escrever, usando a sintaxe apropriada, as regras da gramática da linguagem. Além disso também pode-se especificar outras ações do analisador de acordo com as necessidades do usuário.

3 Compilação e Uso

Para facilitar a compilação do projeto, foi disponibilizado um arquivo Makefile. Nele, há cinco alvos: **clean**, limpa a compilação do projeto; **lex**, compila o analisador léxico; **cup**, compila o analisador sintático; **all**, compila ambos os analisadores e **run** executa os analisadores.

Além disso, para configurar corretamente o projeto, é preciso incluir os arquivos necessários às ferramentas JFLex e CUP.

Ao executar os analisadores, o programa fonte passado como entrada é analisado léxica e sintaticamente. Na saída, o programa fonte é impresso até onde o analisador léxico é bem sucedido, dessa forma, ao ocorrer um error léxico, o mesmo é registrado no ponto do programa onde ocorreu.

Caso a análise léxica seja completada sem erros, o analisador sintático imprimirá, em caso de sucesso, todas as produções gramaticais utilizadas no programa e uma mensagem indicando a conclusão das analises.

Além disso, em caso de exito, todos os tokens presentes no programa fonte serão impressos em um arquivo de texto no diretório do projeto.

4 Testes e Resultados

A fim de ilustrar o uso dos analisadores desenvolvidos, nessa seção é mostrado um exemplo de teste.

4.1 input.txt

- Entrada

```
1  program cond ;
2  v : int ;
3  v1 : real ;
4  k : int ;
5  begin
6          read(v,v1);
7          write( ok );
8          if k > 0 then
9             k := 0;
10 end
```

- Saída
  - Tokens

```
1  <program, ><id,cond><scolon, >
2  <id,v><colon, ><type,int><scolon, >
3  <id,v1><colon, ><type,real><scolon, >
4  <id,k><colon, ><type,int><scolon, >
5  <begin, >
6  <read, ><lfparen, ><id,v><comma, ><id,v1><rtparen, ><scolon, >
7  <write, ><lfparen, ><id,ok><rtparen, ><scolon, >
8  <if, ><id,k><relop,>><num,0><then, >
9  <id,k><assign,:=><num,0><scolon, >
10 <end, >
```

– Análise

```
program -> program ID decl_list compound_stmt
decl_list -> decl_list ; decl
decl -> decl_var
dcl_var -> ident_list : type
indent_list -> id
type -> integer
decl -> decl_var
dcl_var -> ident_list : type
indent_list -> id
type -> real
compound_stmt -> begin stmt_list end
stmt_list -> stmt_list ; stmt
stmt_list -> stmt_list ; stmt
stmt_list -> stmt_list ; stmt
stmt_list ->  stmt
stmt -> read_stmt
stmt -> write_stmt
stmt -> if_stmt
assign_stmt -> id := expr
expr -> Simple_expr
Simple_expr -> term

Accepted
```

5 Código Fonte

Todo o código fonte pode ser obtido em um repositório do GitHub [1].

5.1 tp1.lex

```
package lexsyn;

import java_cup.runtime.*;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.System.*;

%%
%cup
//%class lexAn
//%standalone
%{
    String EXIT_FILE = "tokens.txt";
    FileWriter fw = new FileWriter(EXIT_FILE);
    BufferedWriter bw = new BufferedWriter(fw);

%}

%init{

%init}

%initthrow{
```

[1] https://github.com/MarcelHMendes

```
25      IOException
26  %initthrow}
27
28  %eof{
29      bw.close();
30      fw.close();
31  %eof}
32
33  %eofthrow{
34      IOException
35  %eofthrow}
36
37
38  %%
39
40  [ \t\r\f] { /*eliminando espacos em branco*/
41      System.out.print(yytext());
42  }
43
44  [\n] {
45      System.out.print(yytext());
46     bw.write(yytext());
47  }
48
49  (program) {
50    System.out.print(yytext());
51    bw.write("<"+ "program, >");
52    return new Symbol(sym.PROGRAM);
53
54  }
55
56  (begin) {
57      System.out.print(yytext());
58     bw.write("<"+ "begin, >");
59     return new Symbol(sym.BEGIN);
60  }
61
62  (end) {
63      System.out.print(yytext());
64     bw.write("<"+ "end, >");
65     return new Symbol(sym.END);
66
67  }
68
69  (procedure) {
70          System.out.print(yytext());
71     bw.write("<"+ "procedure, >");
72     return new Symbol(sym.PROCEDURE);
73  }
74
75  ";" {
76    System.out.print(yytext());
77    bw.write("<"+ "scolon, >");
78    return new Symbol(sym.SCOLON);
79
80  }
81
82  ":" {
83          System.out.print(yytext());
84          bw.write("<"+"colon, >");
```

```
85            return new Symbol(sym.COLON);
86
87  }
88
89  "," {
90          System.out.print(yytext());
91          bw.write("<"+"comma, >");
92           return new Symbol(sym.COMMA);
93
94  }
95
96  "(" {
97    System.out.print(yytext());
98    bw.write("<"+ "lfparen, >");
99    return new Symbol(sym.LFPAREN);
100
101 }
102
103 ")" {
104   System.out.print(yytext());
105   bw.write("<"+ "rtparen, >");
106   return new Symbol(sym.RTPAREN);
107 }
108
109 (int) {
110   System.out.print(yytext());
111   bw.write("<" + "type,"+ yytext()+">");
112   return new Symbol(sym.INT);
113 }
114
115 (real) {
116   System.out.print(yytext());
117   bw.write("<" + "type,"+ yytext()+">");
118   return new Symbol(sym.REAL);
119 }
120
121 (boolean) {
122   System.out.print(yytext());
123   bw.write("<" + "type,"+ yytext()+">");
124   return new Symbol(sym.BOOLEAN);
125
126 }
127
128 (char) {
129   System.out.print(yytext());
130   bw.write("<" + "type,"+ yytext()+">");
131   return new Symbol(sym.CHAR);
132 }
133
134 ":=" {
135   System.out.print(yytext());
136   bw.write("<"+ "assign," + yytext()+">");
137   return new Symbol(sym.ASSIGN);
138 }
139
140 /*(while) {
141   System.out.print(yytext());
142   bw.write("<"+ "while, >");
143   return new Symbol(sym.WHILE);
144 }*/
```

```
145
146 (if) {
147   System.out.print(yytext());
148   bw.write("<" +"if, >" );
149   return new Symbol(sym.IF);
150 }
151
152 (else) {
153   System.out.print(yytext());
154   bw.write("<"+"else, >");
155   return new Symbol(sym.ELSE);
156
157 }
158
159 (then) {
160   System.out.print(yytext());
161   bw.write("<"+"then, >");
162   return new Symbol(sym.THEN);
163 }
164
165 (repeat) {
166   System.out.print(yytext());
167   bw.write("<"+ "repeat, >");
168   return new Symbol(sym.REPEAT);
169 }
170
171 (until) {
172   System.out.print(yytext());
173   bw.write("<"+"until, >");
174   return new Symbol(sym.UNTIL);
175 }
176
177 (read) {
178   System.out.print(yytext());
179   bw.write("<"+"read, >");
180   return new Symbol(sym.READ);
181 }
182 (write) {
183   System.out.print(yytext());
184   bw.write("<"+"write, >");
185   return new Symbol(sym.WRITE);
186 }
187
188 (false) {
189   System.out.print(yytext());
190   bw.write("<"+"bool,"+yytext()+">");
191   return new Symbol(sym.FALSE);
192 }
193
194 (true) {
195   System.out.print(yytext());
196   bw.write("<"+"bool,"+yytext()+">");
197   return new Symbol(sym.TRUE);
198
199 }
200
201 (value) {
202   System.out.print(yytext());
203   bw.write("<"+"value, >");
204   return new Symbol(sym.VALUE);
```

```
205 }
206
207 (reference) {
208   System.out.print(yytext());
209   bw.write("<"+"reference, >");
210   return new Symbol(sym.REFERENCE);
211 }
212
213 (not) {
214   System.out.print(yytext());
215   bw.write("<"+"not, >");
216   return new Symbol(sym.NOT);
217
218 }
219
220 (mod) {
221   System.out.print(yytext());
222   bw.write("<"+"mulop,"+yytext()+">");
223   return new Symbol(sym.MOD);
224 }
225
226 (and) {
227   System.out.print(yytext());
228   bw.write("<"+"mulop,"+yytext()+">");
229   return new Symbol(sym.AND);
230 }
231
232 (or) {
233   System.out.print(yytext());
234   bw.write("<"+"addop,"+yytext()+">");
235   return new Symbol(sym.OR);
236 }
237
238
239 [A-Za-z][A-za-z0-9]* {
240     System.out.print(yytext());
241    bw.write("<"+"id,"+yytext()+">");
242    return new Symbol(sym.ID);
243 }
244
245 "<" {
246     System.out.print(yytext());
247    bw.write("<"+"relop,"+ yytext()+">");
248    return new Symbol(sym.LT);
249 }
250
251 ">" {
252   System.out.print(yytext());
253   bw.write("<"+"relop,"+ yytext()+">");
254   return new Symbol(sym.GT);
255 }
256
257 "<=" {
258   System.out.print(yytext());
259   bw.write("<"+ "relop," + yytext()+ ">");
260   return new Symbol(sym.LE);
261 }
262 ">=" {
263   System.out.print(yytext());
264   bw.write("<" + "relop,"+yytext()+">");
```

```
265     return new Symbol(sym.GE);
266   }
267
268   "!=" {
269     System.out.print(yytext());
270     bw.write("<"+ "relop," + yytext()+ ">");
271     return new Symbol(sym.NQ);
272
273   }
274   "=" {
275
276     System.out.print(yytext());
277     bw.write("<"+ "relop," + yytext()+ ">");
278     return new Symbol(sym.EQ);
279   }
280
281   "*" {
282     System.out.print(yytext());
283     bw.write("<" + "mulop,"+yytext()+">");
284     return new Symbol(sym.MUL);
285   }
286
287   "/" {
288     System.out.print(yytext());
289     bw.write("<"+"mulop,"+yytext()+">");
290     return new Symbol(sym.DIV);
291
292   }
293
294
295   "+" {
296     System.out.print(yytext());
297     bw.write("<"+"addop,"+yytext()+">");
298     return new Symbol(sym.PLUS);
299
300   }
301
302   "-" {
303     System.out.print(yytext());
304     bw.write("<"+"addop,"+yytext()+">");
305     return new Symbol(sym.MINUS);
306   }
307
308
309
310   [+|-]?[0-9]*[.][0-9]+([E|e][+|-]?[0-9]+)? { //Real
311     System.out.print(yytext());
312     bw.write("<" + "num,"+ yytext()+">");
313     return new Symbol(sym.REALT, new Double(yytext()));
314   }
315
316   [+|-]?[0-9]+([E|e][+|-]?[0-9]+)? { //inteiro
317       System.out.print(yytext());
318      bw.write("<"+"num,"+yytext()+">");
319      return new Symbol(sym.INTT, new Integer(yytext()));
320   }
321
322   . {
323       System.out.println(" Invalid character "+yytext());
324       System.exit(1);
```

```
325 }
```

5.2 tp1.cup

```
1  package lexsyn;
2
3  import java_cup.runtime.*;
4
5
6  parser code {:
7          public static void main(String args[]) throws Exception {
8                  System.out.println("------------------");
9                  parser myParser = new parser(new Yylex(System.in));
10                 myParser.parse();
11                 System.out.print("\nAccepted");
12         }
13 :};
14 terminal LFPAREN, RTPAREN, SCOLON, BEGIN, END, PROGRAM;
15 terminal INTT, CHART, REALT;
16 terminal INT,REAL,CHAR,BOOLEAN;
17 terminal ASSIGN,COLON,COMMA;
18 terminal IF, ELSE,THEN,UNTIL,REPEAT;
19 terminal READ,WRITE;
20 terminal ID,VALUE,REFERENCE,PROCEDURE;
21 terminal NOT,FALSE,TRUE;
22
23 terminal LT,LE,GT,GE,EQ,NQ;
24 terminal PLUS, MINUS, MUL, DIV,OR,MOD,AND;
25
26 non terminal program, decl_list, decl, dcl_var, indent_list,type;
27 non terminal tipo_retornado,corpo,id_return,espec_parametros;
28 non terminal lista_de_parametros,parametro,modo,compound_stmt,stmt_list;
29 non terminal assign_stmt, if_stmt, cond,repeat_stmt, read_stmt,stmt;
30 non terminal write_stmt,expr_list,expr, Simple_expr, term,factor_a;
31 non terminal factor, function_ref_par,Simple_variable_or_proc;
32 non terminal constant, boolean_constant,dcl_proc,variable;
33
34 precedence left PLUS, MINUS, MUL, DIV, ELSE,MOD;
35
36
37 /*programI ::= program:p {:
38         System.out.println("-----------");
39         String regras = new String("programI -> program\n" + p);
40         System.out.print(regras);
41 :};*/
42 program ::= PROGRAM ID SCOLON decl_list:dl compound_stmt:cs {:
43         System.out.println("-----------");
44         String regras = new String("program -> program ID decl_list compound_stmt\n" + dl + cs);
45         //String regras = new String("programI -> program\n" + p);
46         System.out.print(regras);
47 :};
48 decl_list ::= decl_list:dl SCOLON  decl:dc {:
49                 String RESULTADO = new String("decl_list -> decl_list ; decl\n" + dl+ dc);
50         :}
51         | decl:dc {:
52                 String RESULTADO = new String("decl_list -> decl\n" + dc);
53         :};
54 decl ::= dcl_var:dv {:
55                 String RESULTADO = new String("decl -> decl_var\n" + dv);
```

```
56          :}
57          | dcl_proc:dp {:
58                  String RESULTADO = new String("decl -> decl_proc\n" + dp);
59          :};
60  dcl_var ::= indent_list:il COLON type:t {:
61          String RESULTADO = new String("dcl_var -> ident_list : type\n"+il+t);
62  :};
63   indent_list ::= indent_list:il COMMA ID {:
64                  String RESULTADO =  new String("indent_list ->   : id\n" + il);
65          :}
66          |         ID {:
67                          String RESULTADO = new String("indent_list -> id\n");
68          :};
69  type ::= INT {:
70                  String RESULTADO = new String("type -> integer\n");
71          :}
72          | REAL {:
73                  String RESULTADO = new String("type -> real\n");
74          :}
75          | BOOLEAN {:
76                  String RESULTADO = new String("type -> boolean\n");
77          :}
78          | CHAR {:
79                  String RESULTADO = new String("type -> char\n");
80          :};
81  dcl_proc ::= tipo_retornado:tr PROCEDURE ID espec_parametros:ep corpo:c {:
82                  String RESULTADO = new String("dcl_proc -> tipo_retornado procedure id espec_parametr
83          :};
84
85  tipo_retornado ::= INT {:
86                  String RESULTADO = new String("type -> integer\n");
87          :}
88          | REAL {:
89                  String RESULTADO = new String("type -> real\n");
90          :}
91          | BOOLEAN {:
92                  String RESULTADO = new String("type -> boolean\n");
93          :}
94          | CHAR {:
95                  String RESULTADO = new String("type -> char\n");
96          :}
97          | /* vazio */ {:
98                          String RESULTADO = new String("tipo_retornado -> \n\n");
99          :};
100 corpo ::= COLON decl_list:dl SCOLON compound_stmt:cs id_return:ir {:
101                 String RESULTADO = new String("corpo -> : decl_list ; compound_stmt id_return\n" + dl
102         :}
103         | /* vazio */ {:
104                 String RESULTADO = new String("corpo -> \n\n");
105         :};
106 id_return ::= ID {:
107                 String RESULTADO = new String("id_return -> id\n");
108         :}
109         |         /* vazio */ {:
110                 String RESULTADO = new String("id_return -> \n\n");
111         :};
112
113 espec_parametros ::= LFPAREN lista_de_parametros:lp RTPAREN {:
114                 String RESULTADO = new String("espec_parametros -> ( lista_de_parametros )\n"+ lp);
115         :};
```

```
116 lista_de_parametros ::= parametro:p {:
117                 String RESULTADO = new String("lista_de_parametros -> parametro\n"+ p);
118         :}
119         | lista_de_parametros:lp COMMA parametro:p {:
120                 String RESULTADO = new String("lista_de_parametros -> lista_de_parametros , parametro
121         :};
122 parametro ::= modo:m type:t COLON ID {:
123                 String RESULTADO = new String("parametro -> modo type : id\n"+m+t);
124         :};
125 modo ::= VALUE {:
126                 String RESULTADO = new String("modo -> value\n");
127         :}
128         | REFERENCE {:
129                 String RESULTADO = new String("modo -> reference\n");
130         :};
131 compound_stmt ::= BEGIN stmt_list:sl END {:
132                 String RESULTADO = new String("compound_stmt -> begin stmt_list end\n"+sl);
133         :};
134 stmt_list ::= stmt_list:sl SCOLON stmt:s {:
135                 String RESULTADO = new String("stmt_list -> stmt_list ; stmt\n"+sl+s);
136         :}
137         | stmt:s {:
138                 String RESULTADO = new String("stmt_list -> stmt\n"+s);
139         :}
140         | /*vazio*/ {:
141                 String RESULTADO = new String("stmt_list -> \n\n");
142         :};
143
144 stmt ::= assign_stmt:as {:
145                 String RESULTADO = new String("stmt -> assign_stmt\n"+as);
146         :}
147         | if_stmt:is {:
148                 String RESULTADO = new String("stmt -> if_stmt\n"+is);
149         :}
150         | repeat_stmt:rs {:
151                 String RESULTADO = new String("stmt -> repeat_stmt\n"+ rs);
152         :}
153         | read_stmt:rds {:
154                 String RESULTADO = new String("stmt -> read_stmt\n"+rds);
155         :}
156         | write_stmt:ws {:
157                 String RESULTADO = new String("stmt -> write_stmt\n"+ ws);
158         :}
159         | compound_stmt:cs {:
160                 String RESULTADO = new String("stmt -> compound_stmt\n"+cs);
161         :}
162         | function_ref_par:frp {:
163                 String RESULTADO = new String("stmt -> function_ref_par\n"+frp);
164         :};
165 assign_stmt ::= ID ASSIGN expr:e {:
166                 String RESULTADO = new String("assign_stmt -> id := expr\n"+e);
167         :};
168 if_stmt ::= IF cond:c THEN stmt:s {:
169                 String RESULTADO = new String("if_stmt -> if cond then stmt\n"+c+s);
170         :}
171         | IF cond:c THEN stmt:s1 ELSE stmt:s2 {:
172                         String RESULTADO = new String("if_stmt -> if cond then stmt else stmt\n"+c+s1
173         :};
174 cond ::= expr:e {:
175                 String RESULTADO = new String("cond -> expr\n"+e);
```

```
176         :};
177 repeat_stmt ::= REPEAT stmt_list:st UNTIL expr:e {:
178                 String RESULTADO = new String("repeat_stmt -> repeat stmt_list until expr\n"+st+e);
179         :};
180 read_stmt ::= READ LFPAREN indent_list:il RTPAREN {:
181                 String RESULTADO = new String("read_stmt -> read(  )\n"+il);
182         :};
183 write_stmt ::= WRITE LFPAREN expr_list:el RTPAREN        {:
184                 String RESULTADO = new String("write_stmt -> ( expr_list )\n"+el);
185         :};
186 expr_list ::= expr:e {:
187                 String RESULTADO = new String("expr_list -> expr\n"+e);
188         :}
189         | expr_list:el COMMA expr:e {:
190                 String RESULTADO = new String("expr_list -> expr_list , expr\n"+el+e);
191         :};
192 expr ::= Simple_expr:se {:
193                 String RESULTADO = new String("expr -> Simple_expr\n"+se);
194         :}
195         | Simple_expr:se1 LE Simple_expr:se2 {:
196                 String RESULTADO = new String("expr -> Simple_expr LE Simple_expr \n"+se1+se2);
197         :}
198         | Simple_expr:se1 LT Simple_expr:se2 {:
199                 String RESULTADO = new String("expr -> Simple_expr LT Simple_expr \n"+se1+se2);
200         :}
201         | Simple_expr:se1 GE Simple_expr:se2 {:
202                 String RESULTADO = new String("expr -> Simple_expr GE Simple_expr \n"+se1+se2);
203         :}
204         | Simple_expr:se1 GT Simple_expr:se2 {:
205                 String RESULTADO = new String("expr -> Simple_expr GT Simple_expr \n"+se1+se2);
206         :}
207         | Simple_expr:se1 EQ Simple_expr:se2 {:
208                 String RESULTADO = new String("expr -> Simple_expr EQ  Simple_expr \n"+se1+se2);
209         :}
210         | Simple_expr:se1 NQ Simple_expr:se2 {:
211                 String RESULTADO = new String("expr -> Simple_expr NQ Simple_expr \n"+se1+se2);
212         :};
213
214 Simple_expr ::= term:t {:
215                         String RESULTADO = new String("Simple_expr -> term\n"+t);
216         :}
217         | Simple_expr:se PLUS term:t {:
218                 String RESULTADO = new String("Simple_expr -> Simple_expr + term"+se+t);
219         :}
220         | Simple_expr:se MINUS term:t {:
221                 String RESULTADO = new String("Simple_expr -> Simple_expr - term"+se+t);
222         :}
223         | Simple_expr:se OR term:t {:
224                 String RESULTADO = new String("Simple_expr -> Simple_expr or term"+se+t);
225         :};
226 term ::= factor_a:fa {:
227                 String RESULTADO = new String("term -> factor_a\n"+fa);
228         :}
229         | term:t MUL factor_a:fa {:
230                 String RESULTADO = new String("term -> term * factor_a\n"+t+fa);
231         :}
232         | term:t DIV factor_a:fa {:
233                 String RESULTADO = new String("term -> term / factor_a\n"+t+fa);
234         :}
235         | term:t AND factor_a:fa {:
```

```
236                     String RESULTADO = new String("term -> term and factor_a\n"+t+fa);
237             :}
238             | term:t MOD factor_a:fa {:
239                     String RESULTADO = new String("term -> term mod factor_a\n"+t+fa);
240             :};
241
242     factor_a ::= MINUS factor:f {:
243             String RESULTADO = new String("factor_a -> - factor\n"+f);
244     :}
245     | factor:f {:
246             String RESULTADO = new String("factor_a -> factor\n"+f);
247     :};
248
249     factor ::= ID {:
250             String RESULTADO = new String("factor -> id");
251     :}
252     | constant:c {:
253             String RESULTADO = new String("factor -> constant\n");
254     :}
255     | LFPAREN expr:e RTPAREN {:
256             String RESULTADO = new String("factor -> ( expr )\n"+e);
257     :}
258     | NOT factor:f {:
259             String RESULTADO = new String("factor -> not factor\n"+f);
260     :}
261     | function_ref_par:frp {:
262             String RESULTADO = new String("factor -> function_ref_par\n"+frp);
263     :};
264
265     Simple_variable_or_proc ::= ID {:
266             String RESULTADO = new String("Simple_variable_or_proc -> id\n");
267     :};
268
269     function_ref_par ::= variable:v LFPAREN expr_list:el RTPAREN {:
270             String RESULTADO = new String("function_ref_par -> varialbe (expr_list)"+v+el);
271     :};
272     variable ::= Simple_variable_or_proc {:
273             String RESULTADO = new String("variable -> Simple_variable_or_proc\n");
274     :};
275
276     constant ::= INTT {:
277             String RESULTADO = new String("constant -> integer_constant\n");
278     :}
279     | REALT {:
280             String RESULTADO = new String("constant -> real_constant\n");
281     :}
282     | boolean_constant:bc {:
283             String RESULTADO = new String("constant -> boolean_constant\n"+bc);
284     :}
285     | CHART {:
286             String RESULTADO = new String("constant -> char_constant\n");
287     :};
288
289     boolean_constant ::= FALSE {:
290             String RESULTADO = new String("boolean_constant -> false\n");
291     :}
292     | TRUE {:
293             String RESULTADO = new String("boolean_constant -> true\n");
294     :};
```

5.3 Makefile

```
all: lex cup

.PHONY: lex
lex:
	@  jflex tp1.lex

.PHONY: cup
cup:
	@ java -jar java_cup/java-cup-11b.jar -interface -parser parser *.cup
	@ javac -d . -cp java_cup/java-cup-11b-runtime.jar *.java

.PHONY: run
run:
	@ java -cp java_cup/java-cup-11b-runtime.jar:. lexsyn.parser

.PHONY: clean
clean:
	@ rm  *.java
```

6 Referências

- Aho, A.V.; Sethi, R.; Ullman, J.D. Compilers Principles, Techniques, and Tools, Addison Wesley, 1986.

- Andrew W Appel. Modern Compiler Implementation in Java. Cambridge University Press, 1998. ISBN: 0-521-58388-8.