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

Compiladores I - DCC053
Trabalho Prático 1
Análise Léxica e Sintática LALR

Hugo Araújo de Sousa
(2013007463)

2º Semestre de 2017

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 organizado em tokens, que são sequências significativas para a linguagem em questão. Com esses tokens, pode-se contruir 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.

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

$$
\begin{array}{rcl}
\textit{program} & \rightarrow & \textit{block} \\
\textit{block} & \rightarrow & \texttt{\{}\ \textit{decls}\ \textit{stmts}\ \texttt{\}} \\
\textit{decls} & \rightarrow & \textit{decls}\ \textit{decl} \mid \epsilon \\
\textit{decl} & \rightarrow & \textit{type}\ \textbf{id}\ \texttt{;} \\
\textit{type} & \rightarrow & \textbf{int} \mid \textbf{char} \mid \textbf{bool} \mid \textbf{float} \\
\textit{stmts} & \rightarrow & \textit{stmts}\ \textit{stmt} \mid \epsilon \\
\textit{stmt} & \rightarrow & \textbf{id}\ \texttt{=}\ \textit{expr}\ \texttt{;} \\
 & \mid & \textbf{if}\ \texttt{(}\ \textit{rel}\ \texttt{)}\ \textit{stmt} \\
 & \mid & \textbf{if}\ \texttt{(}\ \textit{rel}\ \texttt{)}\ \textit{stmt}\ \textbf{else}\ \textit{stmt} \\
 & \mid & \textbf{while}\ \texttt{(}\ \textit{rel}\ \texttt{)}\ \textit{stmt} \\
 & \mid & \textit{block} \\
\textit{rel} & \rightarrow & \textit{expr} < \textit{expr} \mid \textit{expr} <= \textit{expr} \mid \textit{expr} >= \textit{expr} \mid \\
 & & \quad \textit{expr} > \textit{expr} \mid \textit{expr} \\
\textit{expr} & \rightarrow & \textit{expr} + \textit{term} \mid \textit{expr} - \textit{term} \mid \textit{term} \\
\textit{term} & \rightarrow & \textit{term} * \textit{unary} \mid \textit{term} / \textit{unary} \mid \textit{unary} \\
\textit{unary} & \rightarrow & - \textit{unary} \mid \textit{factor} \\
\textit{factor} & \rightarrow & \textbf{num} \mid \textbf{real}
\end{array}
$$

Figura 1: 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 JLex e Cup, que funcionam de forma coordenada.

Para gerar analisadores léxicos, a ferramenta JLex 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 JLex. 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 necessário incluir os arquivos necessários às ferramentas JLex e CUP, como mostrado na Figura 2.

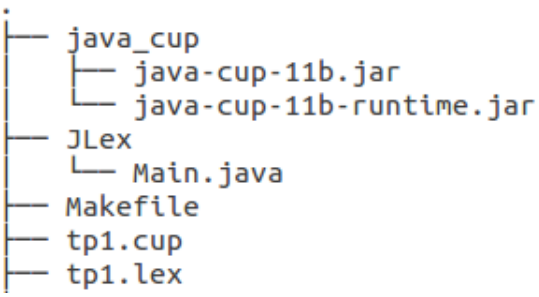

Figura 2: Estrutura do diretório do projeto.

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 sucesso.

Além disso, em caso de sucesso, 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 ilustar o uso dos analisadores desenvolvidos, nessa seção são mostrados dois exemplos de teste.

4.1 input1.txt

- Entrada

```
1  {
2          int weight; int group; int charge; int distance;
3          distance = 2300;
4          weight = 4000;
5          if (60) group = 5;
6          else {
7                  int test;
8                  test = 2000;
9                  group = 1500 / 15;
10                 charge = 40 + 3 * 2300 / 1000;
11                 test = 1;
12         }
13 }
```

- Saída

– Tokens

```
<lcbrack, >
<type, "int"><id, "weight"><semi, ><type, "int"><id, "group"><semi, >
        <type, "int"><id, "charge"><semi, ><type, "int"><id, "distance"><semi, >
<id, "distance"><assign, "="><num, 2300><semi, >
<id, "weight"><assign, "="><num, 4000><semi, >
<if, ><lparen, ><num, 60><rparen, ><id, "group"><assign, "="><num, 5><semi, >
<else, ><lcbrack, >
<type, "int"><id, "test"><semi, >
<id, "test"><assign, "="><num, 2000><semi, >
<id, "group"><assign, "="><num, 1500><binop, "/"><num, 15><semi, >
<id, "charge"><assign, "="><num, 40><binop, "+"><num, 3><binop, "*"><num, 2300>
        <binop, "/"><num, 1000><semi, >
<id, "test"><assign, "="><num, 1><semi, >
<rcbrack, >
<rcbrack, >
```

– Análise

```
----------------------
{
        int weight; int group; int charge; int distance;
        distance = 2300;
        weight = 4000;
        if (60) group = 5;
        else {
                int test;
                test = 2000;
                group = 1500 / 15;
                charge = 40 + 3 * 2300 / 1000;
                test = 1;
        }
}

----------------------
program -> block
block -> { decls stmts }
decls -> decls decl
decls -> decls decl
decls -> decls decl
decls -> decls decl
decls ->

decl -> type id;
type -> int

decl -> type id;
type -> int

decl -> type id;
type -> int

decl -> type id;
type -> int

stmts -> stmts stmt
stmts -> stmts stmt
stmts -> stmts stmt
stmts ->

```

```
42  stmt -> id = expr;
43  expr -> term
44  term -> unary
45  unary -> factor
46  factor -> num
47
48  stmt -> id = expr;
49  expr -> term
50  term -> unary
51  unary -> factor
52  factor -> num
53
54  stmt -> if ( rel ) stmt else stmt
55  rel -> expr
56  expr -> term
57  term -> unary
58  unary -> factor
59  factor -> num
60
61  stmt -> id = expr;
62  expr -> term
63  term -> unary
64  unary -> factor
65  factor -> num
66
67  stmt -> block
68  block -> { decls stmts }
69  decls -> decls decl
70  decls ->
71
72  decl -> type id;
73  type -> int
74
75  stmts -> stmts stmt
76  stmts -> stmts stmt
77  stmts -> stmts stmt
78  stmts -> stmts stmt
79  stmts ->
80
81  stmt -> id = expr;
82  expr -> term
83  term -> unary
84  unary -> factor
85  factor -> num
86
87  stmt -> id = expr;
88  expr -> term
89  term -> term / unary
90  term -> unary
91  unary -> factor
92  factor -> num
93
94  unary -> factor
95  factor -> num
96
97  stmt -> id = expr;
98  expr -> expr + term
99  expr -> term
100 term -> unary
101 unary -> factor
```

```
factor -> num

term -> term / unary
term -> term * unary
term -> unary
unary -> factor
factor -> num

unary -> factor
factor -> num

unary -> factor
factor -> num

stmt -> id = expr;
expr -> term
term -> unary
unary -> factor
factor -> num

Program accepted.
```

4.2 input2.txt

- Entrada

```
{
        int i; int j; int k; bool b;
        i = 4 * 5-3 * -10 / 50;
        j = i * 88;
        k = i* j/ k;
        b = true;
        while (b)
                k = 4 * 3;
}
```

- Saída
  - Tokens

```

```

  - Análise

```
----------------------
{
        int i; int j; int k; bool b;
        i = 4 * 5-3instead expected token classes are [MINUS, MUL, DIV]
Makefile:16: recipe for target 'run' failed
```

5 Código Fonte

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

5.1 tp1.lex

[1] https://github.com/ha2398/compiladores1-tps

```
1  package lexsyn;
2
3  import java_cup.runtime.Symbol;
4  import java.io.BufferedWriter;
5  import java.io.FileWriter;
6  import java.io.IOException;
7
8  %%
9  %cup
10
11 %{
12         String TOKENS_FILENAME = "tokens.txt";
13
14         FileWriter fw = null;
15         BufferedWriter bw =null;
16 %}
17
18 %init{
19         fw = new FileWriter(TOKENS_FILENAME);
20         bw = new BufferedWriter(fw);
21 %init}
22
23 %initthrow{
24         IOException
25 %initthrow}
26
27 %eof{
28         bw.close();
29         fw.close();
30 %eof}
31
32 %eofthrow{
33         IOException
34 %eofthrow}
35
36 %%
37
38 [ \t\r\f] { /* ignore white space. */
39         System.out.print(yytext());
40 }
41
42 [\n] {
43         System.out.print(yytext());
44         bw.write(yytext());
45 }
46
47 "{" {
48         System.out.print(yytext());
49         bw.write("<" + "lcbrack, >");
50         return new Symbol(sym.LCBRACK);
51 }
52
53 "}" {
54         System.out.print(yytext());
55         bw.write("<" + "rcbrack, >");
56         return new Symbol(sym.RCBRACK);
57 }
58
59 ";" {
```

```
60          System.out.print(yytext());
61          bw.write("<" + "semi, >");
62          return new Symbol(sym.SEMI);
63  }
64
65  "(" {
66          System.out.print(yytext());
67          bw.write("<" + "lparen, >");
68          return new Symbol(sym.LPAREN);
69  }
70
71  ")" {
72          System.out.print(yytext());
73          bw.write("<" + "rparen, >");
74          return new Symbol(sym.RPAREN);
75  }
76
77  (int) {
78          System.out.print(yytext());
79          bw.write("<" + "type, \"" + yytext() + "\">");
80          return new Symbol(sym.INTT);
81  }
82
83  (char) {
84          System.out.print(yytext());
85          bw.write("<" + "type, \"" + yytext() + "\">");
86          return new Symbol(sym.CHART);
87  }
88
89  (bool) {
90          System.out.print(yytext());
91          bw.write("<" + "type, \"" + yytext() + "\">");
92          return new Symbol(sym.BOOLT);
93  }
94
95  (float) {
96          System.out.print(yytext());
97          bw.write("<" + "type, \"" + yytext() + "\">");
98          return new Symbol(sym.FLOATT);
99  }
100
101 "=" {
102         System.out.print(yytext());
103         bw.write("<" + "assign, \"" + yytext() + "\">");
104         return new Symbol(sym.ASSIGN);
105 }
106
107 (while) {
108         System.out.print(yytext());
109         bw.write("<" + "while, >");
110         return new Symbol(sym.WHILE);
111 }
112
113 (if) {
114         System.out.print(yytext());
115         bw.write("<" + "if, >");
116         return new Symbol(sym.IF);
117 }
118
119 (else) {
```

```
120         System.out.print(yytext());
121         bw.write("<" + "else, >");
122         return new Symbol(sym.ELSE);
123 }
124
125 [A-Za-z][A-Za-z0-9]* {
126         System.out.print(yytext());
127         bw.write("<" + "id, \"" + yytext() + "\">");
128         return new Symbol(sym.ID);
129 }
130
131 "<" {
132         System.out.print(yytext());
133         bw.write("<" + "relop, \"" + yytext() + "\">");
134         return new Symbol(sym.LT);
135 }
136
137 "<=" {
138         System.out.print(yytext());
139         bw.write("<" + "relop, \"" + yytext() + "\">");
140         return new Symbol(sym.LE);
141 }
142
143 ">=" {
144         System.out.print(yytext());
145         bw.write("<" + "relop, \"" + yytext() + "\">");
146         return new Symbol(sym.GE);
147 }
148
149 ">" {
150         System.out.print(yytext());
151         bw.write("<" + "relop, \"" + yytext() + "\">");
152         return new Symbol(sym.GT);
153 }
154
155 "+" {
156         System.out.print(yytext());
157         bw.write("<" + "binop, \"" + yytext() + "\">");
158         return new Symbol(sym.PLUS);
159 }
160
161 "-" {
162         System.out.print(yytext());
163         bw.write("<" + "binop, \"" + yytext() + "\">");
164         return new Symbol(sym.MINUS);
165 }
166
167 "*" {
168         System.out.print(yytext());
169         bw.write("<" + "binop, \"" + yytext() + "\">");
170         return new Symbol(sym.MUL);
171 }
172
173 "/" {
174         System.out.print(yytext());
175         bw.write("<" + "binop, \"" + yytext() + "\">");
176         return new Symbol(sym.DIV);
177 }
178
179 [+|-]?[0-9]*[.][0-9]+([E|e][+|-]?[0-9]+)? {
```

```
180         System.out.print(yytext());
181         bw.write("<" + "num, " + yytext() + ">");
182         return new Symbol(sym.REALN, new Double(yytext()));
183 }
184
185 [+|-]?[0-9]+([E|e][+|-]?[0-9]+)? {
186         System.out.print(yytext());
187         bw.write("<" + "num, " + yytext() + ">");
188         return new Symbol(sym.INTN, new Integer(yytext()));
189 }
190
191 . {
192         System.out.println("\n\n[ERROR] Illegal character: "+yytext());
193         System.exit(1);
194 }
```

5.2 tp1.cup

```
1  package lexsyn;
2
3  import java_cup.runtime.*;
4
5  parser code {:
6          public static void main(String args[]) throws Exception {
7                  System.out.println("----------------------");
8                  parser myParser = new parser(new Yylex(System.in));
9                  myParser.parse();
10                 System.out.print("Program accepted.");
11         }
12 :}
13
14 terminal LCBRACK, RCBRACK, SEMI, LPAREN, RPAREN;
15 terminal INTT, CHART, BOOLT, FLOATT;
16 terminal ASSIGN;
17 terminal WHILE, IF, ELSE;
18
19 terminal INTN;
20 terminal REALN;
21
22 terminal ID;
23
24 terminal LT, LE, GT, GE;
25 terminal PLUS, MINUS, MUL, DIV;
26
27 non terminal program, block, decls, decl, type, stmts, stmt, rel, expr, term;
28 non terminal unary, factor;
29
30 precedence left PLUS, MINUS, MUL, DIV, ELSE;
31
32 program        ::= block:b {:
33                                 System.out.println("\n----------------------");
34                                 String rules = new String("program -> block\n" + b);
35                                 System.out.print(rules);
36                         :}
37         ;
38 block        ::= LCBRACK decls:ds stmts:ss RCBRACK {:
39                                 RESULT = new String("block -> { decls stmts }\n" + ds + ss);
40                         :}
41         ;
```

```
42  decls        ::= decls:ds decl:d {:
43                                   RESULT = new String("decls -> decls decl\n" + ds + d);
44                               :}
45          |                 /* empty */ {:
46                                   RESULT = new String("decls -> \n\n");
47                               :}
48          ;
49  decl         ::= type:t ID SEMI {:
50                                   RESULT = new String("decl -> type id;\n" + t);
51                               :}
52          ;
53  type        ::= INTT {:
54                                   RESULT = new String("type -> int\n\n");
55                               :}
56          |                 CHART {:
57                                   RESULT = new String("type -> char\n\n");
58                               :}
59          |                 BOOLT {:
60                                   RESULT = new String("type -> bool\n\n");
61                               :}
62          |                 FLOATT {:
63                                   RESULT = new String("type -> float\n\n");
64                               :}
65          ;
66  stmts        ::= stmts:ss stmt:s {:
67                                   RESULT = new String("stmts -> stmts stmt\n" + ss + s);
68                               :}
69          |                  /* empty */ {:
70                                   RESULT = new String("stmts -> \n\n");
71                               :}
72          ;
73  stmt         ::= ID ASSIGN expr:e SEMI {:
74                                   RESULT = new String("stmt -> id = expr;\n" + e);
75                               :}
76          |                   IF LPAREN rel:r RPAREN stmt:s {:
77                                   RESULT = new String("stmt -> if ( rel ) stmt\n" + r + s);
78                               :}
79          |                   IF LPAREN rel:r RPAREN stmt:s1 ELSE stmt:s2 {:
80                                   RESULT = new String("stmt -> if ( rel ) stmt else stmt\n" +
81                                   r + s1 + s2);
82                               :}
83          |                 WHILE LPAREN rel:r RPAREN stmt:s {:
84                                   RESULT = new String("stmt -> while ( rel ) stmt\n" + r + s);
85                               :}
86          |                 block:b {:
87                                   RESULT = new String("stmt -> block\n" + b);
88                               :}
89          ;
90  rel          ::= expr:e1 LT expr:e2 {:
91                                   RESULT = new String("rel -> expr < expr\n" + e1 + e2);
92                               :}
93          |                   expr:e1 LE expr:e2 {:
94                                   RESULT = new String("rel -> expr <= expr\n" + e1 + e2);
95                               :}
96          |                   expr:e1 GT expr:e2 {:
97                                   RESULT = new String("rel -> expr > expr\n" + e1 + e2);
98                               :}
99          |                   expr:e1 GE expr:e2 {:
100                                  RESULT = new String("rel -> expr >= expr\n" + e1 + e2);
101                              :}
```

```
102         |                expr:e {:
103                                 RESULT = new String("rel -> expr\n" + e);
104                             :}
105         ;
106 expr          ::= expr:e PLUS term:t {:
107                                 RESULT = new String("expr -> expr + term\n" + e + t);
108                             :}
109         |                 expr:e MINUS term:t {:
110                                 RESULT = new String("expr -> expr - term\n" + e + t);
111                             :}
112         |                 term:t {:
113                                 RESULT = new String("expr -> term\n" + t);
114                             :}
115         ;
116 term          ::= term:t MUL unary:u {:
117                                 RESULT = new String("term -> term * unary\n" + t + u);
118                             :}
119          |                  term:t DIV unary:u {:
120                                 RESULT = new String("term -> term / unary\n" + t + u);
121                             :}
122          |                  unary:u {:
123                                 RESULT = new String("term -> unary\n" + u);
124                             :}
125          ;
126 unary          ::= MINUS unary:u {:
127                                 RESULT = new String("unary -> - unary\n" + u);
128                             :}
129         |                 factor:f {:
130                                 RESULT = new String("unary -> factor\n" + f);
131                             :}
132         ;
133 factor         ::= INTN:n {:
134                                 RESULT = new String("factor -> num\n\n");
135                             :}
136         |                 REALN:n {:
137                                 RESULT = new String("factor -> real\n\n");
138                             :}
139         ;
```

5.3 Makefile

```
1  all: lex cup
2
3  .PHONY: lex
4  lex:
5          @ javac JLex/Main.java
6          @ java JLex.Main *.lex
7          @ mv *.lex.java Yylex.java
8
9  .PHONY: cup
10 cup:
11         @ java -jar java_cup/java-cup-11b.jar -interface -parser parser *.cup
12         @ javac -d . -cp java_cup/java-cup-11b-runtime.jar *.java
13
14 .PHONY: run
15 run:
16         @ java -cp java_cup/java-cup-11b-runtime.jar:. lexsyn.parser
17
18 .PHONY: clean
```

```
19 clean:
20         @ rm -f JLex/*.class *.lex.java
21         @ rm -f *.java
22         @ rm -rf lexsyn
23         @ rm -rf *.txt
```

6 Referências

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