

语法分析程序的设计与实现

班级:2017211301 学号:2017211137 姓名: 王亚宇

2019 年 12 月 2 日

目录

目录	1
1 实验内容	2
1.1 实验内容	2
1.2 实验要求	2
2 实验环境	2
2.1 操作系统	2
2.2 编译环境	2
3 实验过程	2
3.1 编写词法分析程序	2
3.1.1 lex.l	2
3.1.2 lex.y	3
3.1.3 生成词法分析程序	4
3.1.4 LARA 项目簇与 DFA	4
3.1.5 分析过程	10
3.1.5.1 测试用例	10
3.1.5.2 分析过程	10
4 Appendices	18
4.1 lex.l	18
4.2 lex.y	19

1 实验内容

1.1 实验内容

编写语法分析程序，实现对算数表达式的语法分析。要求所分析算数表达式由如下文法产生。

$$\begin{aligned} E &\rightarrow E + T \mid E - T \mid T \\ T &\rightarrow T * F \mid T / F \mid F \\ F &\rightarrow (E) \mid num \end{aligned}$$

1.2 实验要求

在对输入的算数表达式进行分析的过程中，一次输出所采用的产生式。利用 YACC 自动生成语法分析程序，调用 LEX 自动生成的词法分析程序。

2 实验环境

2.1 操作系统

Ubuntu 18.04 LTS

2.2 编译环境

g++ (Ubuntu 7.4.0-1ubuntu1 18.04.1) 7.4.0 flex 2.6.4 bison (GNU Bison) 3.0.4

3 实验过程

3.1 编写词法分析程序

3.1.1 lex.l

```
1  %{
2  #include <bits/stdc++.h>
3  #include "y.tab.hh"
4  extern "C"{
5  int yywrap(void);
```

```

6  int yylex(void);
7  }
8  using namespace std;
9  %}
10
11  digit          [0-9]
12  exponent       ([Ee] [+ -]? {digit}+)
13
14  %%
15
16  [-/+*()]       { yylval.char_value = yytext[0];
   ↪ return yytext[0]; }
17  0|([1-9]{digit}*) { yylval.double_value = atoi(yytext);
   ↪ return CONSTANT; }
18  {digit}+{exponent}? { yylval.double_value = atof(yytext);
   ↪ return CONSTANT; }
19  {digit}*"."{digit}+{exponent}? { yylval.double_value = atof(yytext);
   ↪ return CONSTANT; }
20  {digit}+"."{digit}*{exponent}?? { yylval.double_value = atof(yytext);
   ↪ return CONSTANT; }
21
22  %%
23
24  int yywrap(void) {
25      return 1;
26  }

```

使用 lex 识别运算符和数字

3.1.2 lex.y

```

1  %union {
2      char char_value;
3      double double_value;
4  }
5
6  %start E

```

```

7
8 %token <double_value> CONSTANT
9 %type <double_value> E T F
10
11 %left <char_value> '+' '-' '*' '/'
12 %token <char_value> '(' ')'

```

重写 yylval 类型并给予所有文法符号定义类型以进行值传递和输出。

```

1 E : E '+' T { $$ = $1 + $3 ;}
2   | E '-' T { $$ = $1 - $3 ;}
3   | T      { $$ = $1      ;}
4   ;
5
6 T : T '*' F { $$ = $1 * $3 ;}
7   | T '/' F { $$ = $1 / $3 ;}
8   | F      { $$ = $1      ;}
9   ;
10
11 F : '(' E ')' { $$ = $2      ;}
12   | CONSTANT { $$ = $1      ;}
13   ;

```

声明产生式，并进行数值运算。

3.1.3 生成词法分析程序

```

flex lex.l ; bison -vd --debug lex.y -o y.tab.cc ; g++ lex.yy.c
↪ y.tab.cc -o lex

```

3.1.4 LARA 项目簇与 DFA

Grammar

```
0 $accept: E $end
```

```

1 E: E '+' T
2   | E '-' T
3   | T

4 T: T '*' F
5   | T '/' F
6   | F

7 F: '(' E ')'
8   | CONSTANT

```

Terminals, with rules where they appear

```

$end (0) 0
'(' (40) 7
')' (41) 7
'*' (42) 4
'+' (43) 1
 '-' (45) 2
 '/' (47) 5
error (256)
CONSTANT (258) 8

```

Nonterminals, with rules where they appear

```

$accept (10)
  on left: 0
E (11)
  on left: 1 2 3, on right: 0 1 2 7
T (12)
  on left: 4 5 6, on right: 1 2 3 4 5
F (13)
  on left: 7 8, on right: 4 5 6

```

State 0

0 \$accept: . E \$end

CONSTANT shift, and go to state 1

'(' shift, and go to state 2

E go to state 3

T go to state 4

F go to state 5

State 1

8 F: CONSTANT .

\$default reduce using rule 8 (F)

State 2

7 F: '(' . E ')'

CONSTANT shift, and go to state 1

'(' shift, and go to state 2

E go to state 6

T go to state 4

F go to state 5

State 3

0 \$accept: E . \$end

1 E: E . '+' T

2 | E . '-' T

```
$end  shift, and go to state 7
'+'   shift, and go to state 8
'-'   shift, and go to state 9
```

State 4

```
3 E: T .
4 T: T . '*' F
5  | T . '/' F

'*'   shift, and go to state 10
'/'   shift, and go to state 11

$default  reduce using rule 3 (E)
```

State 5

```
6 T: F .

$default  reduce using rule 6 (T)
```

State 6

```
1 E: E . '+' T
2  | E . '-' T
7 F: '(' E . ')'

'+'   shift, and go to state 8
'-'   shift, and go to state 9
')'   shift, and go to state 12
```

State 7

0 \$accept: E \$end .

\$default accept

State 8

1 E: E '+' . T

CONSTANT shift, and go to state 1

'(' shift, and go to state 2

T go to state 13

F go to state 5

State 9

2 E: E '-' . T

CONSTANT shift, and go to state 1

'(' shift, and go to state 2

T go to state 14

F go to state 5

State 10

4 T: T '*' . F

CONSTANT shift, and go to state 1

'(' shift, and go to state 2

F go to state 15

State 11

5 T: T '/' . F

CONSTANT shift, and go to state 1

'(' shift, and go to state 2

F go to state 16

State 12

7 F: '(' E ')' .

\$default reduce using rule 7 (F)

State 13

1 E: E '+' T .

4 T: T . '*' F

5 | T . '/' F

'*' shift, and go to state 10

'/' shift, and go to state 11

\$default reduce using rule 1 (E)

State 14

2 E: E '-' T .

4 T: T . '*' F

5 | T . '/' F

'*' shift, and go to state 10

'/' shift, and go to state 11

```
$default  reduce using rule 2 (E)
```

State 15

```
4 T: T '*' F .
```

```
$default  reduce using rule 4 (T)
```

State 16

```
5 T: T '/' F .
```

```
$default  reduce using rule 5 (T)
```

3.1.5 分析过程

3.1.5.1 测试用例

```
1 (3+5/2)*(2.4/3)-(1-2)
```

3.1.5.2 分析过程

```
1 Starting parse
2 Entering state 0
3 Reading a token: Next token is token '(' ('(')
4 Shifting token '(' ('(')
5 Entering state 2
6 Reading a token: Next token is token CONSTANT (3)
7 Shifting token CONSTANT (3)
8 Entering state 1
9 Reducing stack by rule 8 (line 55):
10 $1 = token CONSTANT (3)
11 -> $$ = nterm F (3)
```

```

12 Stack now 0 2
13 Entering state 5
14 Reducing stack by rule 6 (line 51):
15     $1 = nterm F (3)
16     -> $$ = nterm T (3)
17 Stack now 0 2
18 Entering state 4
19 Reading a token: Next token is token '+' ('+')
20 Reducing stack by rule 3 (line 46):
21     $1 = nterm T (3)
22     -> $$ = nterm E (3)
23 Stack now 0 2
24 Entering state 6
25 Next token is token '+' ('+')
26 Shifting token '+' ('+')
27 Entering state 8
28 Reading a token: Next token is token CONSTANT (5)
29 Shifting token CONSTANT (5)
30 Entering state 1
31 Reducing stack by rule 8 (line 55):
32     $1 = token CONSTANT (5)
33     -> $$ = nterm F (5)
34 Stack now 0 2 6 8
35 Entering state 5
36 Reducing stack by rule 6 (line 51):
37     $1 = nterm F (5)
38     -> $$ = nterm T (5)
39 Stack now 0 2 6 8
40 Entering state 13
41 Reading a token: Next token is token '/' ('/')
42 Shifting token '/' ('/')
43 Entering state 11
44 Reading a token: Next token is token CONSTANT (2)
45 Shifting token CONSTANT (2)
46 Entering state 1
47 Reducing stack by rule 8 (line 55):

```

```

48     $1 = token CONSTANT (2)
49 -> $$ = nterm F (2)
50 Stack now 0 2 6 8 13 11
51 Entering state 16
52 Reducing stack by rule 5 (line 50):
53     $1 = nterm T (5)
54     $2 = token '/' ('/')
55     $3 = nterm F (2)
56 -> $$ = nterm T (2.500000)
57 Stack now 0 2 6 8
58 Entering state 13
59 Reading a token: Next token is token ')' ('')
60 Reducing stack by rule 1 (line 44):
61     $1 = nterm E (3)
62     $2 = token '+' ('+')
63     $3 = nterm T (2.500000)
64 -> $$ = nterm E (5.500000)
65 Stack now 0 2
66 Entering state 6
67 Next token is token ')' ('')
68 Shifting token ')' ('')
69 Entering state 12
70 Reducing stack by rule 7 (line 54):
71     $1 = token '(' ('(')
72     $2 = nterm E (5.500000)
73     $3 = token ')' ('')
74 -> $$ = nterm F (5.500000)
75 Stack now 0
76 Entering state 5
77 Reducing stack by rule 6 (line 51):
78     $1 = nterm F (5.500000)
79 -> $$ = nterm T (5.500000)
80 Stack now 0
81 Entering state 4
82 Reading a token: Next token is token '*' ('*')
83 Shifting token '*' ('*')

```

```

84 Entering state 10
85 Reading a token: Next token is token '(' ('(')
86 Shifting token '(' ('(')
87 Entering state 2
88 Reading a token: Next token is token CONSTANT (2.400000)
89 Shifting token CONSTANT (2.400000)
90 Entering state 1
91 Reducing stack by rule 8 (line 55):
92     $1 = token CONSTANT (2.400000)
93     -> $$ = nterm F (2.400000)
94 Stack now 0 4 10 2
95 Entering state 5
96 Reducing stack by rule 6 (line 51):
97     $1 = nterm F (2.400000)
98     -> $$ = nterm T (2.400000)
99 Stack now 0 4 10 2
100 Entering state 4
101 Reading a token: Next token is token '/' ('/')
102 Shifting token '/' ('/')
103 Entering state 11
104 Reading a token: Next token is token CONSTANT (3)
105 Shifting token CONSTANT (3)
106 Entering state 1
107 Reducing stack by rule 8 (line 55):
108     $1 = token CONSTANT (3)
109     -> $$ = nterm F (3)
110 Stack now 0 4 10 2 4 11
111 Entering state 16
112 Reducing stack by rule 5 (line 50):
113     $1 = nterm T (2.400000)
114     $2 = token '/' ('/')
115     $3 = nterm F (3)
116     -> $$ = nterm T (0.800000)
117 Stack now 0 4 10 2
118 Entering state 4
119 Reading a token: Next token is token ')' (')')

```

```

120 Reducing stack by rule 3 (line 46):
121     $1 = nterm T (0.800000)
122 -> $$ = nterm E (0.800000)
123 Stack now 0 4 10 2
124 Entering state 6
125 Next token is token ')' ('')
126 Shifting token ')' ('')
127 Entering state 12
128 Reducing stack by rule 7 (line 54):
129     $1 = token '(' ('(')
130     $2 = nterm E (0.800000)
131     $3 = token ')' ('')
132 -> $$ = nterm F (0.800000)
133 Stack now 0 4 10
134 Entering state 15
135 Reducing stack by rule 4 (line 49):
136     $1 = nterm T (5.500000)
137     $2 = token '*' ('*')
138     $3 = nterm F (0.800000)
139 -> $$ = nterm T (4.400000)
140 Stack now 0
141 Entering state 4
142 Reading a token: Next token is token '-' ('-')
143 Reducing stack by rule 3 (line 46):
144     $1 = nterm T (4.400000)
145 -> $$ = nterm E (4.400000)
146 Stack now 0
147 Entering state 3
148 Next token is token '-' ('-')
149 Shifting token '-' ('-')
150 Entering state 9
151 Reading a token: Next token is token '(' ('(')
152 Shifting token '(' ('(')
153 Entering state 2
154 Reading a token: Next token is token CONSTANT (1)
155 Shifting token CONSTANT (1)

```

```

156 Entering state 1
157 Reducing stack by rule 8 (line 55):
158     $1 = token CONSTANT (1)
159 -> $$ = nterm F (1)
160 Stack now 0 3 9 2
161 Entering state 5
162 Reducing stack by rule 6 (line 51):
163     $1 = nterm F (1)
164 -> $$ = nterm T (1)
165 Stack now 0 3 9 2
166 Entering state 4
167 Reading a token: Next token is token '-' ('-')
168 Reducing stack by rule 3 (line 46):
169     $1 = nterm T (1)
170 -> $$ = nterm E (1)
171 Stack now 0 3 9 2
172 Entering state 6
173 Next token is token '-' ('-')
174 Shifting token '-' ('-')
175 Entering state 9
176 Reading a token: Next token is token CONSTANT (2)
177 Shifting token CONSTANT (2)
178 Entering state 1
179 Reducing stack by rule 8 (line 55):
180     $1 = token CONSTANT (2)
181 -> $$ = nterm F (2)
182 Stack now 0 3 9 2 6 9
183 Entering state 5
184 Reducing stack by rule 6 (line 51):
185     $1 = nterm F (2)
186 -> $$ = nterm T (2)
187 Stack now 0 3 9 2 6 9
188 Entering state 14
189 Reading a token: Next token is token ')' (')')
190 Reducing stack by rule 2 (line 45):
191     $1 = nterm E (1)

```

```

192     $2 = token '-' ('-')
193     $3 = nterm T (2)
194 -> $$ = nterm E (-1)
195 Stack now 0 3 9 2
196 Entering state 6
197 Next token is token ')' (')')
198 Shifting token ')' (')')
199 Entering state 12
200 Reducing stack by rule 7 (line 54):
201     $1 = token '(' ('(')
202     $2 = nterm E (-1)
203     $3 = token ')' (')')
204 -> $$ = nterm F (-1)
205 Stack now 0 3 9
206 Entering state 5
207 Reducing stack by rule 6 (line 51):
208     $1 = nterm F (-1)
209 -> $$ = nterm T (-1)
210 Stack now 0 3 9
211 Entering state 14
212 Reading a token:
213 Now at end of input.
214 Reducing stack by rule 2 (line 45):
215     $1 = nterm E (4.400000)
216     $2 = token '-' ('-')
217     $3 = nterm T (-1)
218 -> $$ = nterm E (5.400000)
219 Stack now 0
220 Entering state 3
221 Now at end of input.
222 Shifting token $end ()
223 Entering state 7
224 Stack now 0 3 7
225 Cleanup: popping token $end ()
226 Cleanup: popping nterm E (5.400000)

```

最后计算得表达式值为 5.4 与实际吻合极好

×
–
□
Programming Mode ▾

(3+5/2)*(2.4/3)-(1-2)

=

5.4

5.4

Decimal ▾

FF₁₆ 256₁₀

0000 0000 0000 0000 0000 0000 0000 0000

63 47 32

0000 0000 0000 0000 0000 0000 0000 0000

31 15 0

↓n

↑n

.

x ▾

(

)

< ▾

> ▾

á

C

D

E

F

÷

mod

ones

twos

|x|

8

9

A

B

×

AND

NOT

√

x^y

x⁻¹

4

5

6

7

–

OR

C

log

log₂

int

0

1

2

3

+

XOR

=

fact

x!

frac

4 Appendices

4.1 lex.l

```
1  %{
2  #include <bits/stdc++.h>
3  #include "y.tab.hh"
4  extern "C"{
5  int yywrap(void);
6  int yylex(void);
7  }
8  using namespace std;
9  %}
10
11 digit                [0-9]
12 exponent             ([Ee] [+-]?{digit}+)
13
14 %%
15
16 [-/+*()]            { yylval.char_value = yytext[0];
   ↪ return yytext[0]; }
17 0|([1-9]{digit}*?)  { yylval.double_value = atoi(yytext);
   ↪ return CONSTANT; }
18 {digit}+{exponent}? { yylval.double_value = atof(yytext);
   ↪ return CONSTANT; }
19 {digit}*"."{digit}+{exponent}? { yylval.double_value = atof(yytext);
   ↪ return CONSTANT; }
20 {digit}+"."{digit}*{exponent}?? { yylval.double_value = atof(yytext);
   ↪ return CONSTANT; }
21
22 %%
23
24 int yywrap(void) {
25     return 1;
26 }
```

4.2 lex.y

```
1  %{
2      #include <bits/stdc++.h>
3      #define YYDEBUG 1
4      #define eps 1e-5
5      extern "C" {
6          void yyerror(const char *){};
7          extern int yylex(void);
8          extern char *yytext;
9          extern int yydebug;
10     }
11
12     using namespace std;
13 }%
14
15 %union {
16     char char_value;
17     double double_value;
18 }
19
20 %start E
21
22 %token <double_value> CONSTANT
23 %type <double_value> E T F
24
25 %left <char_value> '+' '-' '*' '/'
26 %token <char_value> '(' ')'
27
28 %printer {
29     int r = round($$);
30     if( abs(r-$$)<=eps ){
31         fprintf (yyo, "%d", r);
32     }
33     else{
34         fprintf (yyo, "%lf", $$);
```

```

35     }
36 } <double_value>
37
38 %printer {
39     fprintf (yyo, "'%c'", $$);
40 } <char_value>
41
42 %%
43
44 E : E '+' T { $$ = $1 + $3 ;}
45   | E '-' T { $$ = $1 - $3 ;}
46   | T       { $$ = $1       ;}
47   ;
48
49 T : T '*' F { $$ = $1 * $3 ;}
50   | T '/' F { $$ = $1 / $3 ;}
51   | F       { $$ = $1       ;}
52   ;
53
54 F : '(' E ')' { $$ = $2       ;}
55   | CONSTANT { $$ = $1       ;}
56   ;
57
58 %%
59
60 int main() {
61     yydebug = 1;
62     return yyparse();
63 }

```