

算术表达式的合法性判断与求值（上）

2014年10月06日 20:37:37 神奕 阅读量: 7744 更多

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/lisong694767315/article/details/39831571>

在写一个计算器时遇到了一个问题，就是对字符串表示的算术表达式的合法性判断与求值。下面记录一下我的解决方案。

一、问题描述

问题：给定一个字符串，只包含 +, -, *, /, 数字, 小数点, (,)。

要求：(1) 判断该算术表达式是否合法； (2) 如果合法，计算该表达式的值。

二、判断表达式的合法性

相信学过《编译原理》的人都知道，利用里面讲的分析方法可以对源代码进行解析。而算术表达式也是源代码的一部分，所以利用编译方法也可以很容易地判断表达式的合法性。

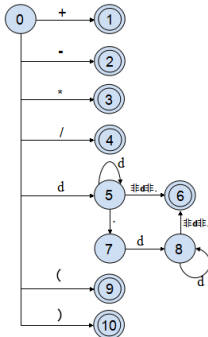
与源代码相比，算术表达式只包含有很少的字符，所以解析起来也简单很多。下面从词法分析和语法分析两个方面来说明。

1) 词法分析

下面先定一下表达式涉及到的单词的种别编码：

单词符号	种别编码
+	1
-	2
*	3
/	4
数字	5
(6
)	7

识别上表所列的单词的状态转换图：



C++实现：

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 #include <utility>
5 using namespace std;
6
7 int word_analysis(vector<pair<string, int>>& word, const string expr)
8 {
9     for(int i=0; i<expr.length(); ++i)
10     {
11         // 如果是 + - * / ( )
12         if(expr[i] == '(' || expr[i] == ')' || expr[i] == '+'
13            || expr[i] == '-' || expr[i] == '*' || expr[i] == '/')
14         {
15             string tmp;
16             tmp.push_back(expr[i]);
17             switch (expr[i])
18             {
19                 case '+':
20                     word.push_back(make_pair(tmp, 1));
21                     break;
22                 case '-':
23                     word.push_back(make_pair(tmp, 2));
24                     break;
25                 case '*':
26                     word.push_back(make_pair(tmp, 3));
27                     break;
28                 case '/':
29                     word.push_back(make_pair(tmp, 4));
30                     break;
31                 case '(':
32                     word.push_back(make_pair(tmp, 6));
33                     break;
34                 case ')':
35                     word.push_back(make_pair(tmp, 7));
36                     break;
37             }
38         }
39         // 如果是数字开头
40         else if(expr[i]>='0' && expr[i]<='9')
41         {
42             string tmp;
43             while(expr[i]>='0' && expr[i]<='9')
44             {
45                 tmp.push_back(expr[i]);
46                 ++i;
47             }
48             if(expr[i] == '.')
49             {
```

```

50         ++i;
51         if(expr[i]>='0' && expr[i]<='9')
52         {
53             tmp.push_back('.');
54             while(expr[i]>='0' && expr[i]<='9')
55             {
56                 tmp.push_back(expr[i]);
57                 ++i;
58             }
59         }
60         else
61         {
62             return -1; // .后面不是数字, 词法错误
63         }
64     }
65     word.push_back(make_pair(tmp, 5));
66     --i;
67 }
68 // 如果以, 开头
69 else
70 {
71     return -1; // 以, 开头, 词法错误
72 }
73 }
74 return 0;
75 }
76
77 int main()
78 {
79     vector<pair<string, int>> word;
80     string expr = "(1.5+5.789)*82-10/2";
81
82     int err_num = word_analysis(word, expr);
83     if (-1 == err_num)
84         cout << "Word Error!" << endl;
85     else
86         cout << "No Word Error!" << endl;
87     return 0;
88 }

```

上面的代码将识别出的单词-种别编码对 (**单词, 种别编码**) 存入一个 `vector<pair<string, int>>` 中。

2) 语法分析

算术表达式的文法 G[E] 如下:

```

E → E+T | E-T | T
T → T*F | T/F | F
F → (E) | d

```

消去非终符号E和T的左递归后, 改写 G[E] 文法如下:

```

E → TE'
E' → +TE' | -TE' | ε
T → FT'
T' → *FT' | /FT' | ε
F → (E) | d

```

可以证明上述无递归文法是 LL(1) 文法, 可以使用 **递归下降分析法**。递归下降分析法是确定的自上而下分析法, 这种分析法要求文法是 LL(1) 文法。它的**基本思想**是: 对文法中的每一个非终结符编写一个函数 (或子程序), 每个函数 (或子程序) 的功能是识别由该非终结符所表示的语法成分。

构造递归下降分析程序时, 每个函数名是相应的非终结符, 函数体是根据规则右部符号串的结构编写:

- 当遇到终结符 a 时, 则编写语句
if (当前读来的输入符号 == a) 读下一个输入符号;
- 当遇到非终结符 A 时, 则编写语句调用 A();
- 当遇到 A->ε 规则时, 则编写语句
if (当前读来的输入符号 不属于 FOLLOW(A)) error();
- 当某个非终结符的规则有多个候选式时, 按 LL(1) 文法的条件能唯一地选择一个候选式进行推导。

所以我们需要求出 **FOLLOW(E')** 和 **FOLLOW(T')**:

FOLLOW(E')= FOLLOW(E)={ }, #

FOLLOW(T')= FOLLOW(T)={+, -, }, #

好了, 下面直接上代码, 在词法分析的基础上进行语法分析:

```

1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <utility>
5  using namespace std;
6
7  vector<pair<string, int>> word;
8  string expr = "(1.5+5.789)*82-10/2";
9  int idx = 0;
10 int sym;
11 int err = 0; // 错误
12
13 void E();
14 void E1();
15 void T();
16 void T1();
17 void F();
18 /*-----词法分析-----*/
19 int word_analysis(vector<pair<string, int>>& word, const string expr)
20 {
21     for(int i=0; i<expr.length(); ++i)
22     {
23         // 如果是 + - * / ( )
24         if(expr[i] == '(' || expr[i] == ')') || expr[i] == '+'
25             || expr[i] == '-' || expr[i] == '*' || expr[i] == '/')
26         {
27             string tmp;

```

```

28         tmp.push_back(expr[i]);
29         switch (expr[i])
30         {
31             case '+':
32                 word.push_back(make_pair(tmp, 1));
33                 break;
34             case '-':
35                 word.push_back(make_pair(tmp, 2));
36                 break;
37             case '*':
38                 word.push_back(make_pair(tmp, 3));
39                 break;
40             case '/':
41                 word.push_back(make_pair(tmp, 4));
42                 break;
43             case '(':
44                 word.push_back(make_pair(tmp, 6));
45                 break;
46             case ')':
47                 word.push_back(make_pair(tmp, 7));
48                 break;
49         }
50     }
51     // 如果是数字开头
52     else if(expr[i]>='0' && expr[i]<='9')
53     {
54         string tmp;
55         while(expr[i]>='0' && expr[i]<='9')
56         {
57             tmp.push_back(expr[i]);
58             ++i;
59         }
60         if(expr[i] == '.')
61         {
62             ++i;
63             if(expr[i]>='0' && expr[i]<='9')
64             {
65                 tmp.push_back('.');
66                 while(expr[i]>='0' && expr[i]<='9')
67                 {
68                     tmp.push_back(expr[i]);
69                     ++i;
70                 }
71             }
72             else
73             {
74                 return -1; // 后面不是数字, 词法错误
75             }
76         }
77         word.push_back(make_pair(tmp, 5));
78         --i;
79     }
80     // 如果以, 开头
81     else
82     {
83         return -1; // 以, 开头, 词法错误
84     }
85 }
86 return 0;
87 }
88 /*-----语法分析-----*/
89 // 读下一单词的种别编码
90 void Next()
91 {
92     if(idx < word.size())
93         sym = word[idx++].second;
94     else
95         sym = 0;
96 }
97
98 //  $E \rightarrow TE'$ 
99 void E()
100 {
101     T();
102     E1();
103 }
104
105 //  $E' \rightarrow +TE' \mid -TE' \mid \varepsilon$ 
106 void E1()
107 {
108     if(sym == 1)
109     {
110         Next();
111         T();
112         E1();
113     }
114     else if(sym == 2)
115     {
116         Next();
117         T();
118         E1();
119     }
120     else if(sym != 7 && sym != 0)
121     {
122         err = -1;
123     }
124 }
125
126 //  $T \rightarrow FT'$ 
127 void T()
128 {
129     F();
130     T1();
131 }
132
133 //  $T' \rightarrow *FT' \mid /FT' \mid \varepsilon$ 
134 void T1()
135 {
136     if(sym == 3)
137     {
138         Next();
139         F();
140         T1();
141     }
142     else if(sym == 4)

```

```

143     {
144         Next();
145         F();
146         T1();
147     }
148     else if(sym != 1 && sym != 2 && sym != 7 && sym != 0)
149     {
150         err = -1;
151     }
152 }
153
154 //  $F \rightarrow (E) \mid d$ 
155 void F()
156 {
157     if(sym == 5)
158     {
159         Next();
160     }
161     else if(sym == 6)
162     {
163         Next();
164         E();
165         if(sym == 7)
166         {
167             Next();
168         }
169         else
170         {
171             err = -1;
172         }
173     }
174     else
175     {
176         err = -1;
177     }
178 }
179
180 int main()
181 {
182     int err_num = word_analysis(word, expr);
183     cout << expr << endl << "Word Analysis:" << endl;
184     if (-1 == err_num)
185     {
186         cout << "Word Error!" << endl;
187     }
188     else
189     {
190         // 测试输出
191         vector<pair<string, int>>::iterator beg = word.begin();
192         for(; beg != word.end(); ++beg)
193             cout << " (" << beg->first << ", " << beg->second << ")" << endl;
194
195         // 词法正确, 进行语法分析
196         Next();
197         E();
198         if (sym == 0 && err == 0) // 注意要判断两个条件
199             cout << "Right Expression." << endl;
200         else
201             cout << "Wrong Expression." << endl;
202     }
203     return 0;
204 }

```

另外, 还有一种更简单的形式, 将文法 G(E) 用扩充BNF表示法进行改写:

```

E → T { + T | - T }
T → F { * F | / F }
F → (E) | d

```

然后对这种变形文法使用递归下降分析法:

```

1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <utility>
5  using namespace std;
6
7  vector<pair<string, int>> word;
8  string expr = "(1.5+5.789)*82-10/2";
9  int idx = 0;
10 int sym;
11 int err = 0; // 错误
12
13 void T();
14 void F();
15
16 /*-----词法分析-----*/
17 int word_analysis(vector<pair<string, int>>& word, const string expr)
18 {
19     for(int i=0; i<expr.length(); ++i)
20     {
21         // 如果是 + - * / ( )
22         if(expr[i] == '(' || expr[i] == ')' || expr[i] == '+'
23            || expr[i] == '-' || expr[i] == '*' || expr[i] == '/')
24         {
25             string tmp;
26             tmp.push_back(expr[i]);
27             switch (expr[i])
28             {
29                 case '+':
30                     word.push_back(make_pair(tmp, 1));
31                     break;
32                 case '-':
33                     word.push_back(make_pair(tmp, 2));
34                     break;
35                 case '*':
36                     word.push_back(make_pair(tmp, 3));
37                     break;
38                 case '/':
39                     word.push_back(make_pair(tmp, 4));
40             }

```

```

39         break;
40         case '(':
41             word.push_back(make_pair(tmp, 6));
42             break;
43         case ')':
44             word.push_back(make_pair(tmp, 7));
45             break;
46     }
47 }
48 // 如果是数字开头
49 else if(expr[i]>='0' && expr[i]<='9')
50 {
51     string tmp;
52     while(expr[i]>='0' && expr[i]<='9')
53     {
54         tmp.push_back(expr[i]);
55         ++i;
56     }
57     if(expr[i] == '.')
58     {
59         ++i;
60         if(expr[i]>='0' && expr[i]<='9')
61         {
62             tmp.push_back('.');
63             while(expr[i]>='0' && expr[i]<='9')
64             {
65                 tmp.push_back(expr[i]);
66                 ++i;
67             }
68         }
69         else
70         {
71             return -1; // 后面不是数字, 词法错误
72         }
73     }
74     word.push_back(make_pair(tmp, 5));
75     --i;
76 }
77 // 如果以, 开头
78 else
79 {
80     return -1; // 以, 开头, 词法错误
81 }
82 }
83 return 0;
84 }
85 /*-----语法分析-----*/
86 // 读下一单词的种别编码
87 void Next()
88 {
89     if(idx < word.size())
90         sym = word[idx++].second;
91     else
92         sym = 0;
93 }
94
95 //  $E \rightarrow T \mid +T \mid -T$ 
96 void E()
97 {
98     T();
99     while(sym == 1 || sym == 2)
100     {
101         Next();
102         T();
103     }
104 }
105
106 //  $T \rightarrow F \mid *F \mid /F$ 
107 void T()
108 {
109     F();
110     while(sym == 3 || sym == 4)
111     {
112         Next();
113         F();
114     }
115 }
116
117 //  $F \rightarrow (E) \mid d$ 
118 void F()
119 {
120     if (sym == 5)
121     {
122         Next();
123     }
124     else if(sym == 6)
125     {
126         Next();
127         E();
128         if (sym == 7)
129         {
130             Next();
131         }
132         else
133         {
134             err = -1;
135         }
136     }
137     else
138     {
139         err = -1;
140     }
141 }
142
143 int main()
144 {
145     int err_num = word_analysis(word, expr);
146     cout << expr << endl << "Word Analysis:" << endl;
147     if (-1 == err_num)
148     {
149         cout << "Word Error!" << endl;
150     }
151     else
152     {
153         // 测试输出

```

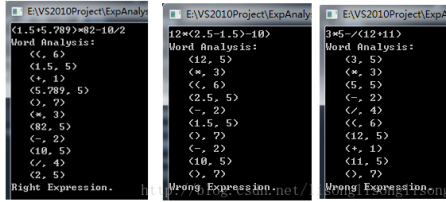
```

154         vector<pair<string, int>>::iterator beg = word.begin();
155         for(;beg!=word.end(); ++beg)
156             cout << " " << beg->first << ", " << beg->second << ")" << endl;
157
158         // 词法正确, 进行语法分析
159         Next();
160         E();
161         if (sym == 0 && err == 0) // 注意要判断两个条件
162             cout << "Right Expression." << endl;
163         else
164             cout << "Wrong Expression." << endl;
165     }
166     return 0;
167 }

```

推荐这种文法形式! 因为基于这种文法形式写程序, 只需要写3个函数 (因为只有3个非终结符), 而且不要求 FOLLOW 集合。

测试结果:



[算术表达式的合法性判断与求值 \(下\) >>](#)