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 阅读数：7744 更多

CSDN 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。

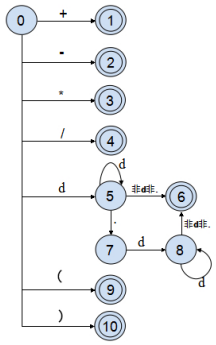
与源代码相比，算术表达式只包含有很少的字符，所以解析起来也简单很多。下面从词法分析和语法分析两个方面来说明。

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：

```
#include <iostream>
#include <vector>
#include <string>
#include <utility>
using namespace std;

int word_analysis(vector<pair<string, int>>& word, const string expr)
{
        for(int i=0; i<expr.length(); ++i)
        {
                // 如果是 + - * / ( )
                if(expr[i] == '(' || expr[i] == ')' || expr[i] == '+'
                        || expr[i] == '-' || expr[i] == '*' || expr[i] == '/')
                {
                        string tmp;
                        tmp.push_back(expr[i]);
                        switch (expr[i])
                        {
                                case '+':
                                        word.push_back(make_pair(tmp, 1));
                                        break;
                                case '-':
                                        word.push_back(make_pair(tmp, 2));
                                        break;
                                case '*':
                                        word.push_back(make_pair(tmp, 3));
                                        break;
                                case '/':
                                        word.push_back(make_pair(tmp, 4));
                                        break;
                                case '(':
                                        word.push_back(make_pair(tmp, 6));
                                        break;
                                case ')':
                                        word.push_back(make_pair(tmp, 7));
                                        break;
                        }
                }
                // 如果是数字开头
                else if(expr[i]>='0' && expr[i]<='9')
                {
                        string tmp;
                        while(expr[i]>='0' && expr[i]<='9')
                        {
                                tmp.push_back(expr[i]);
                                ++i;
                        }
                        if(expr[i] == '.')
                        {
```

```
                                ++i;
                                if(expr[i]>='0' && expr[i]<='9')
                                {
                                        tmp.push_back('.');
                                        while(expr[i]>='0' && expr[i]<='9')
                                        {
                                                tmp.push_back(expr[i]);
                                                ++i;
                                        }
                                }
                                else
                                {
                                        return -1;  // .后面不是数字，词法错误
                                }
                        }
                        word.push_back(make_pair(tmp, 5));
                        --i;
                }
                // 如果以.开头
                else
                {
                        return -1;  // 以.开头，词法错误
                }
        }
        return 0;
}

int main()
{
    vector<pair<string, int>> word;
    string expr = "(1.5+5.789)*82-10/2";

    int err_num = word_analysis(word, expr);
    if (-1 == err_num)
        cout << "Word Error!" << endl;
    else
        cout << "No Word Error!" << endl;
    return 0;
}
```

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

消去非终结符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：

```
#include <iostream>
#include <vector>
#include <string>
#include <utility>
using namespace std;

vector<pair<string, int>> word;
string expr = "(1.5+5.789)*82-10/2";
int idx = 0;
int sym;
int err = 0; // 错误

void E();
void E1();
void T();
void T1();
void F();
/*--------------------------------词法分析-----------------------------*/
int word_analysis(vector<pair<string, int>>& word, const string expr)
{
        for(int i=0; i<expr.length(); ++i)
        {
                // 如果是 + - * / ( )
                if(expr[i] == '(' || expr[i] == ')' || expr[i] == '+'
                        || expr[i] == '-' || expr[i] == '*' || expr[i] == '/')
                {
                        string tmp;
```

```
                        tmp.push_back(expr[i]);
                        switch (expr[i])
                        {
                        case '+':
                                word.push_back(make_pair(tmp, 1));
                                break;
                        case '-':
                                word.push_back(make_pair(tmp, 2));
                                break;
                        case '*':
                                word.push_back(make_pair(tmp, 3));
                                break;
                        case '/':
                                word.push_back(make_pair(tmp, 4));
                                break;
                        case '(':
                                word.push_back(make_pair(tmp, 6));
                                break;
                        case ')':
                                word.push_back(make_pair(tmp, 7));
                                break;
                        }
                }
                // 如果是数字开头
                else if(expr[i]>='0' && expr[i]<='9')
                {
                        string tmp;
                        while(expr[i]>='0' && expr[i]<='9')
                        {
                                tmp.push_back(expr[i]);
                                ++i;
                        }
                        if(expr[i] == '.')
                        {
                                ++i;
                                if(expr[i]>='0' && expr[i]<='9')
                                {
                                        tmp.push_back('.');
                                        while(expr[i]>='0' && expr[i]<='9')
                                        {
                                                tmp.push_back(expr[i]);
                                                ++i;
                                        }
                                }
                                else
                                {
                                        return -1;  // .后面不是数字，词法错误
                                }
                        }
                        word.push_back(make_pair(tmp, 5));
                        --i;
                }
                // 如果以.开头
                else
                {
                        return -1;  // 以.开头，词法错误
                }
        }
        return 0;
}
/*————————————————————————————语法分析————————————————————————————*/
// 读下一单词的种别编码
void Next()
{
        if(idx < word.size())
                sym = word[idx++].second;
        else
                sym = 0;
}

// E → TE'
void E()
{
        T();
        E1();
}

// E' → +TE' | -TE' | ε
void E1()
{
        if(sym == 1)
        {
                Next();
                T();
                E1();
        }
        else if(sym == 2)
        {
                Next();
                T();
                E1();
        }
        else if(sym != 7 && sym != 0)
        {
                err = -1;
        }
}

// T → FT'
void T()
{
        F();
        T1();
}

// T' → *FT' | /FT' | ε
void T1()
{
        if(sym == 3)
        {
                Next();
                F();
                T1();
        }
        else if(sym == 4)
```

```
        {
                Next();
                F();
                T1();
        }
        else if(sym != 1 && sym != 2 && sym != 7 && sym != 0)
        {
                err = -1;
        }
}

// F → (E) | d
void F()
{
        if(sym == 5)
        {
                Next();
        }
        else if(sym == 6)
        {
                Next();
                E();
                if(sym == 7)
                {
                        Next();
                }
                else
                {
                        err = -1;
                }
        }
        else
        {
                err = -1;
        }
}

int main()
{
        int err_num = word_analysis(word, expr);
        cout << expr << endl << "Word Analysis:" << endl;
        if (-1 == err_num)
        {
                cout << "Word Error!" << endl;
        }
        else
        {
                // 测试输出
                vector<pair<string, int>>::iterator beg = word.begin();
                for(;beg!=word.end(); ++beg)
                        cout << "   (" << beg->first << ", " << beg->second << ")" << endl;

                // 词法正确，进行语法分析
                Next();
                E();
                if (sym == 0 && err == 0)  // 注意要判断两个条件
                        cout << "Right Expression." << endl;
                else
                        cout << "Wrong Expression." << endl;
        }
        return 0;
}
```

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：

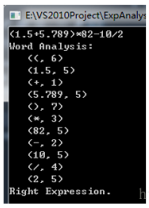
```
#include <iostream>
#include <vector>
#include <string>
#include <utility>
using namespace std;

vector<pair<string, int>> word;
string expr = "(1.5+5.789)*82-10/2";
int idx = 0;
int sym;
int err = 0; // 错误

void T();
void F();
/*-------------------------------词法分析-------------------------------*/
int word_analysis(vector<pair<string, int>>& word, const string expr)
{
        for(int i=0; i<expr.length(); ++i)
        {
                // 如果是 + - * / ( )
                if(expr[i] == '(' || expr[i] == ')' || expr[i] == '+'
                        || expr[i] == '-' || expr[i] == '*' || expr[i] == '/')
                {
                        string tmp;
                        tmp.push_back(expr[i]);
                        switch (expr[i])
                        {
                        case '+':
                                word.push_back(make_pair(tmp, 1));
                                break;
                        case '-':
                                word.push_back(make_pair(tmp, 2));
                                break;
                        case '*':
                                word.push_back(make_pair(tmp, 3));
                                break;
                        case '/':
                                word.push_back(make_pair(tmp, 4));
```

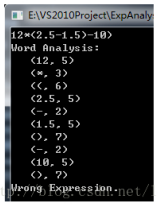
```
                        break;
                case '(':
                        word.push_back(make_pair(tmp, 6));
                        break;
                case ')':
                        word.push_back(make_pair(tmp, 7));
                        break;
                }
        }
        // 如果是数字开头
        else if(expr[i]>='0' && expr[i]<='9')
        {
                string tmp;
                while(expr[i]>='0' && expr[i]<='9')
                {
                        tmp.push_back(expr[i]);
                        ++i;
                }
                if(expr[i] == '.')
                {
                        ++i;
                        if(expr[i]>='0' && expr[i]<='9')
                        {
                                tmp.push_back('.');
                                while(expr[i]>='0' && expr[i]<='9')
                                {
                                        tmp.push_back(expr[i]);
                                        ++i;
                                }
                        }
                        else
                        {
                                return -1;  // .后面不是数字，词法错误
                        }
                }
                word.push_back(make_pair(tmp, 5));
                --i;
        }
        // 如果以.开头
        else
        {
                return -1;  // 以.开头，词法错误
        }
    }
    return 0;
}
/*--------------------------------语法分析--------------------------------*/
// 读下一单词的种别编码
void Next()
{
    if(idx < word.size())
            sym = word[idx++].second;
    else
            sym = 0;
}

// E → T { +T | -T }
void E()
{
    T();
    while(sym == 1 || sym == 2)
    {
            Next();
            T();
    }
}

// T → F { *F | /F }
void T()
{
    F();
    while(sym == 3 || sym == 4)
    {
            Next();
            F();
    }
}

// F → (E) | d
void F()
{
    if (sym == 5)
    {
            Next();
    }
    else if(sym == 6)
    {
            Next();
            E();
            if (sym == 7)
            {
                    Next();
            }
            else
            {
                    err = -1;
            }
    }
    else
    {
            err = -1;
    }
}

int main()
{
    int err_num = word_analysis(word, expr);
    cout << expr << endl << "Word Analysis:" << endl;
    if (-1 == err_num)
    {
            cout << "Word Error!" << endl;
    }
    else
    {
            // 测试输出
```

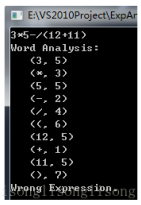
```
154                 vector<pair<string, int>>::iterator beg = word.begin();
155                 for(;beg!=word.end(); ++beg)
156                         cout << "   (" << beg->first << ", " << beg->second << ")" << endl;
157
158                 // 词法正确，进行语法分析
159                 Next();
160                 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），而且不需要求 FOLLOW 集合。**

测试结果：







算术表达式的合法性判断与求值（下）>>