**语义分析程序实验报告**

**班级：2014211302**

**学号：2014211168**

**姓名：周尧棋**

**一、实验要求**

编写语义分析程序，实现对算术表达式的类型检查和求值。要求所分析算术表达式由如下的文法产生。

E->E+T|E-T|T

T->T\*F|T/F|F

F-> (E)|num| num.num

用自底向上的语法制导翻译技术实现对表达式的分析和翻译。

(1)写出满足要求的语法制导定义或翻译方案。

(2)编写分析程序，实现对表达式的类型进行检查和求值，并输出：

1.分析过程中所有产生式。

2.识别出的表达式的类型。

3.识别出的表达式的值。

(3)实验方法：可以选用以下两种方法之一。

1.自己编写分析程序。

2.利用YACC自动生成工具。

**二、程序设计说明**

**1.整体说明**

（1）根据题目所给出的文法构造相应的拓广文法，并求出该文法各非终结符的FIRST、FOLLOW集合；

（2）构造拓广文法的项目集规范族，并构造出识别所有前缀的DFA；

（3）构造文法的LR分析表；

（4）由此构造LR分析程序。

（5）写出满足要求的翻译方案。

（6）实现对表达式的类型进行检查和求值，并输出。

**2.构造拓广文法，并求出该文法各非终结符的FIRST、FOLLOW集合**

首先构造文法的拓广文法G'：

(0)E'->E

(1)E->E+T

(2)E->E-T

(3)E->T

(4)T->T\*F

(5)T->T/F

(6)T->F

(7)F->(E)

(8)F->num

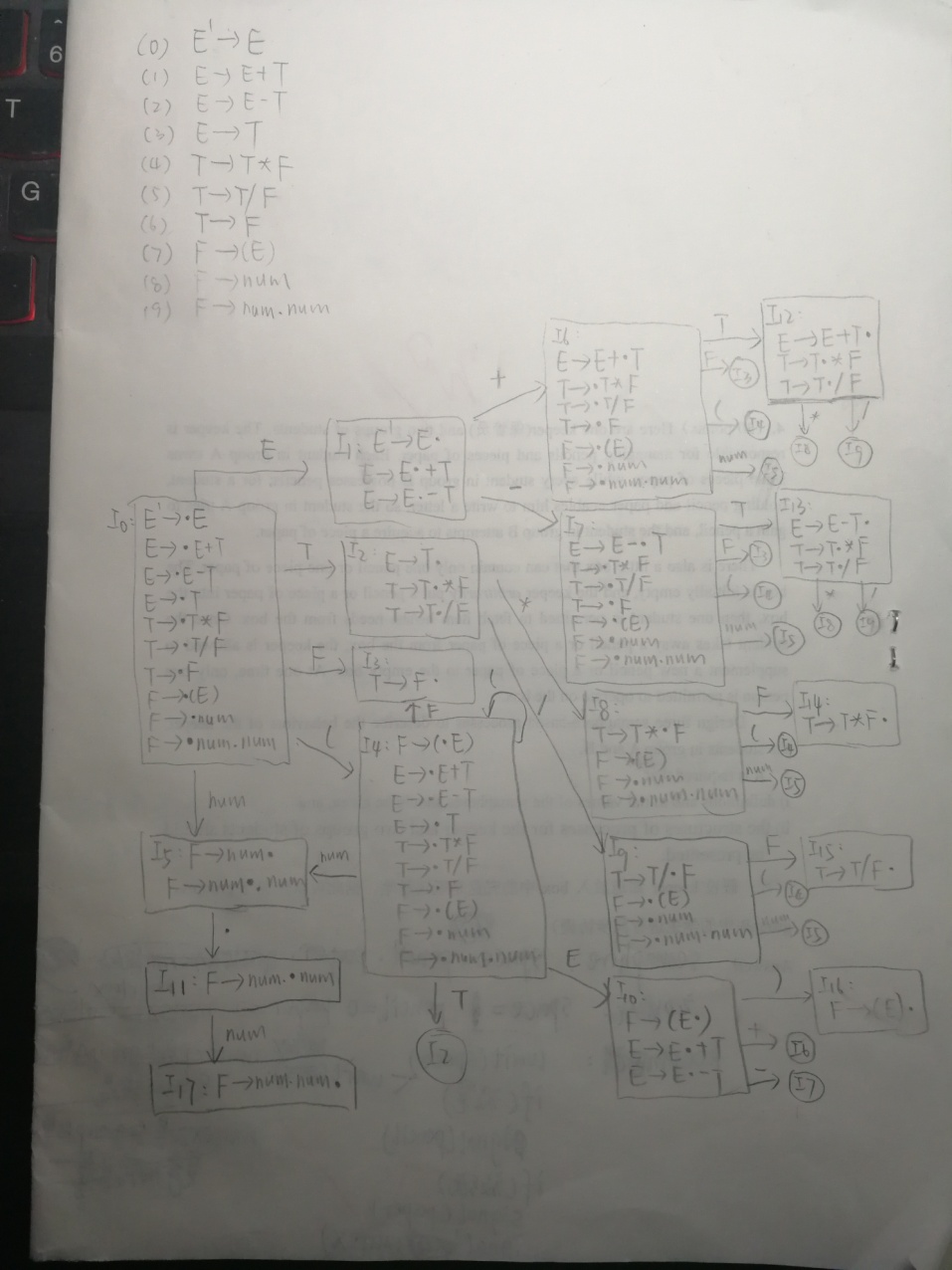
(9)F->num.num

**文法各非终结符的FIRST、FOLLOW集合**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E' | E | T | F |
| FIRST | (, num | (, num | (, num | (, num |
| FOLLOW | $ | $, ), +, - | $, ), +, -, \*, / | $, ), +, -, \*, / |

**3.构造 LR(0)项目集规范族及识别其所有活前缀的DFA**

首先构造ε活前缀的LR(0)有效项目集I0，然后根据go(I,X)转移函数分别求出其他活前缀的LR(0)有效项目集，最终得到LR(0)项目集规范族及识别其所有活前缀的DFA，构造结果如下图：

****

**4.构造LR分析表**

根据书上算法4.6，在构造完文法G'的LR(0)项目集规范族后，按照算法规则进行LR分析表的构造，注意一点的是，若A->α· 属于某个LR(0)项目集Ii，对所有FOLLOW(A)中的符号a，置action[i,a]=R A->α，表示用产生式A->α进行规约。

其次，在LR分析过程中，当出现action表项为空的情况时，会发生错误，在该程序中我引入了六种错误处理方法。

e1：状态0、2、4、6、7、8、9期待(或者 num，但输入为+、-、\*、/或$时，打印错误“缺少运算对象”。

e2：状态0、1、2、4、6、7 、8 、9期待运算对象首字符或运算符号，但输入为) 时，打印错误“括号不匹配”。

e3：状态1、2、10 、11、12期待运算符号或)，但输入是(或num，打印错误“缺少运算符”。

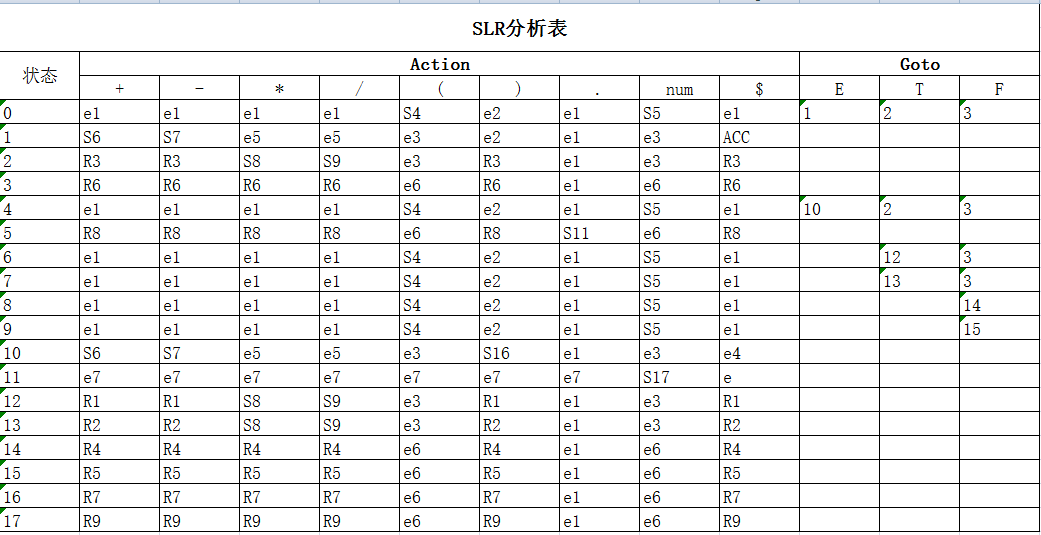
e4：状态10期待运算符号或者)，但输入是$时，打印错误“缺少右括号”。

e5：状态1、10期待+或-号，但输入是\*或/时，打印错误“缺少+/-符号”。

e6：其他归约状态时，输入为(或者num时，打印错误“缺少运算符号”。

e7：小数点后不是数字，打印错误“小数点后不是数字”。

最终得到LR分析表，结果如下图



**5.翻译方案**

(1)E->E’+T{E.val = E’.val + T.val}

{if (E’.type == real || T.type == real) E.type = real; else E.type = integer;}

(2)E->E’-T{E.val = E’.val - T.val}

{if (E’.type == real || T.type == real) E.type = real; else E.type = integer;}

(3)E ->T{E.val = T.val}{E.type = T.type}

(4)T->T’\*F{T.val = T’.val \* F.val}

{if (T’.type == real || F.type == real) T.type = real; else T.type = integer;}

(5)T->T’/F{T.val = T’.val / F.val}

{if (T’.type == real || F.type == real) T.type = real; else T.type = integer;}

(6)T ->F{T.val = F.val}{T.type = F.type}

(7)F ->(E){F.val = E.val}{F.type = E.type}

(8)F ->num{F.val = num.val} {F.type = integer}

(9)F ->num.num{F.val = num.num.val} {F.type = real}

6.分析翻译动作

首先先手动求出算术表达式文法所有活前缀的DFA，由该DFA构造出文法的LR分析表，得到两个二维数组string Action[][]和int Goto[][]。进入程序后，先通过函数Init()初始化文法生成式，将生成式存入容器P中，并将终结符存入容器VT中，将非终结符存入容器VN中，以便下面的操作。利用算法4.3构造LR分析过程，引入了错误处理机制，完成对输入字符串的语法分析。根据上面的翻译方案，在函数Reduce中实现了对表达式的类型进行检查和求值。当需要规约时便调用Reduce函数进行栈中元素的变化，打印栈中状态。

**三、源程序**

#include<iostream>

#include<cctype>

#include<cmath>

#include<cstdlib>

#include<vector>

#include<sstream>

#include<string>

#include<stack>

using namespace std;

vector<char> VN; //非终结符号集合

vector<string> VT; //终结符号集合

vector<string> P; //产生式集合

int State[50]; //状态栈

double Value[50];

int Type[40]; //integer:1 real:2

int top = 0;

//初始化LR分析表中的action表

string Action[20][20] =

{

{"e1","e1","e1","e1","S4","e2","e1","S5","e1"},

{"S6","S7","e5","e5","e3","e2","e1","e3","ACC"},

{"R3","R3","S8","S9","e3","R3","e1","e3","R3"},

{"R6","R6","R6","R6","e6","R6","e1","e6","R6"},

{"e1","e1","e1","e1","S4","e2","e1","S5","e1"},

{"R8","R8","R8","R8","e6","R8","S11","e6","R8"},

{"e1","e1","e1","e1","S4","e2","e1","S5","e1"},

{"e1","e1","e1","e1","S4","e2","e1","S5","e1"},

{"e1","e1","e1","e1","S4","e2","e1","S5","e1"},

{"e1","e1","e1","e1","S4","e2","e1","S5","e1"},

{"S6","S7","e5","e5","e3","S16","e1","e3","e4"},

{"e7","e7","e7","e7","e7","e7","e7","S17","e7"},

{"R1","R1","S8","S9","e3","R1","e1","e3","R1"},

{"R2","R2","S8","S9","e3","R2","e1","e3","R2"},

{"R4","R4","R4","R4","e6","R4","e1","e6","R4"},

{"R5","R5","R5","R5","e6","R5","e1","e6","R5"},

{"R7","R7","R7","R7","e6","R7","e1","e6","R7"},

{"R9","R9","R9","R9","e6","R9","e1","e6","R9"}

};

//初始化LR分析表中的goto表

int Goto[20][4] =

{

{1,2,3},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

{10,2,3},

{-1,-1,-1},

{-1,12,3},

{-1,13,3},

{-1,-1,14},

{-1,-1,15},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

};

void Init()

{

VN.push\_back('E');

VN.push\_back('T');

VN.push\_back('F');

VT.push\_back("+");

VT.push\_back("-");

VT.push\_back("\*");

VT.push\_back("/");

VT.push\_back("(");

VT.push\_back(")");

VT.push\_back(".");

VT.push\_back("num");

P.push\_back("E->E+T");

P.push\_back("E->E-T");

P.push\_back("E->T");

P.push\_back("T->T\*F");

P.push\_back("T->T/F");

P.push\_back("T->F");

P.push\_back("F->(E)");

P.push\_back("F->num");

P.push\_back("F->num.num");

}

int string\_to\_int(string str)

//将string型转换为int型

{

int num;

stringstream stream;

stream << str;

stream >> num;

return num;

}

int findVN(vector<char> F, char c)

//在VN集中寻找非终结符，返回下标

{

vector<char>::iterator it;

for (it = F.begin(); it != F.end(); it ++)

{

if ((\*it) == c)

{

return it - F.begin();

}

}

return -1;

}

int finds(vector<string> F, string c)

//在VT集中寻找终结符，返回下标

{

vector<string>::iterator it;

for (it = F.begin(); it != F.end(); it ++)

{

if ((\*it) == c)

{

return it - F.begin();

}

}

return -1;

}

void Reduce(int chioce)

//翻译方案

{

int i;

double j;

switch (chioce)

{

case 1:

Value[top-2] = Value[top-2] + Value[top];

if (Type[top] == 2 || Type[top-2] == 2)

{

Type[top-2] = 2;

}

else

{

Type[top-2] = 1;

}

break;

case 2:

Value[top-2] = Value[top-2] - Value[top];

if (Type[top] == 2 || Type[top-2] == 2)

{

Type[top-2] = 2;

}

else

{

Type[top-2] = 1;

}

break;

case 3:

Value[top] = Value[top];

Type[top] = Type[top];

break;

case 4:

Value[top-2] = Value[top-2] \* Value[top];

if (Type[top] == 2 || Type[top-2] == 2)

{

Type[top-2] = 2;

}

else

{

Type[top-2] = 1;

}

break;

case 5:

if (Type[top] == 2 || Type[top-2] == 2 || (int)Value[top-2]%(int)Value[top] != 0)

{

Type[top-2] = 2;

}

else

{

Type[top-2] = 1;

}

Value[top-2] = Value[top-2] / Value[top];

break;

case 6:

Value[top] = Value[top];

Type[top] = Type[top];

break;

case 7:

Value[top-2] = Value[top-1];

Type[top-2] = Type[top-1];

break;

case 8:

Value[top] = Value[top];

Type[top] = 1;

break;

case 9:

j = Value[top];

//计算小数点后有多少位

for (i = 0; j >= 1; i ++)

{

j = j / 10;

}

Value[top-2] = Value[top-2] + Value[top] \* pow(0.1,i);

Type[top-2] = 2;

break;

default:

break;

}

}

void output(int choice)

//打印State和Value栈内容

{

int i;

if (choice == 0)

{

for (i = 0; i <= top; i ++)

{

cout << State[i] << " " ;

}

}

else

{

for (i = 0; i <= top; i ++)

{

cout << Value[i] << " " ;

}

}

}

void LR\_analysis()

{

int i, j, m;

int ip = 0;

int length = 0;

int S;

char a;

string tmp;

VT.push\_back("$");

string input;

cout << "输入符号串：" << endl;

cin >> input;

cout << endl;

input += "$";

State[top] = 0; //将状态0压入State栈中

cout << "分析过程为：" << endl;

while(1)

{

S = State[top];

a = input[ip];

tmp.clear();

cout << "State：" ;

output(0);

cout << endl;

cout << "Value：" ;

output(1);

cout << endl << "输入：" << input.substr(ip) << '\t' << "分析动作：";

if (a == '+' || a == '-' || a == '\*' || a == '/' ||

a == '(' || a == ')' || a == '.' || a == '$')

{

length = 1;

}

else

{

j = ip;

length = 0;

i = 0;

while(isdigit(a))

{

i = i \* 10 + a - '0';

length ++;

j ++;

a = input[j];

}

a = 'n';

}

if (a == 'n')

{

tmp = "num";

}

else

{

tmp += a;

}

m = finds(VT,tmp);

if (Action[S][m][0] == 'S')

//移进

{

top ++;

if (tmp == "num")

{

Value[top] = i;

}

else

{

Value[top] = 0;

}

State[top] = string\_to\_int(Action[S][m].substr(1));

ip += length;

cout << "Shift " << Action[S][m].substr(1) << endl << endl;

tmp.clear();

}

else if (Action[S][m][0] == 'R')

//规约

{

int r = string\_to\_int(Action[S][m].substr(1));

int cnt = 0;

Reduce(r);

string str;

str = P[r-1].substr(3);

int k = 0;

string temp1;

//计算A->b中|b|的长度，存入cnt

while (k < str.length())

{

temp1 += str[k];

if (findVN(VN,str[k]) >= 0 || finds(VT,temp1) >= 0)

{

cnt ++;

temp1.clear();

}

k ++;

}

temp1.clear();

//cout << cnt << endl;

//从栈顶弹出|b|个元素

for (int m = 0; m < cnt; m ++)

{

top --;

}

//\_S为栈顶当前状态

int \_S = State[top];

string A = P[r-1].substr(0,1);

//将goto[\_S,A]压入State栈中

int vn = findVN(VN,A[0]);

top ++;

State[top] = Goto[\_S][vn];

//cout << "State[top] = " << State[top] << endl;

//输出A->b

cout << "reduce by ：" << P[r-1] << "\tvalue: " << Value[top];

cout << "\ttype: ";

if (Type[top] == 1)

{

cout << "integer";

}

else

{

cout << "real";

}

cout << endl << endl ;

}

else if (Action[S][m][0] == 'A')

{

cout << "ACC" << "\t表达式的值: " << Value[top];

cout << "\t表达式的类型: ";

if (Type[top] == 1)

{

cout << "integer";

}

else

{

cout << "real";

}

cout << endl;

break;

}

else if (Action[S][m][0] == 'e')

{

cout << "error!" << endl;

if (Action[S][m][1] == '1')

//状态0 2 4 6 7 8 9 期待(或者 num，但输入为+、-、\*、/或$时 ->缺少运算对象

{

top ++;

Value[top] = 0;

State[top] = 5;

cout << "缺少运算对象！将num压入栈" << endl;

}

else if (Action[S][m][1] == '2')

//状态0 1 2 4 6 7 8 9 期待运算对象首字符或运算符号，但输入为) ->括号不匹配

{

ip ++;

cout << "括号不匹配！删掉“）”" << endl;

}

else if (Action[S][m][1] == '3')

//状态1 2 10 11 12期待运算符号或)，但输入是(或num ->缺少运算符

{

Value[top] = '+';

State[top] = 6;

cout << "缺少运算符号！将+压入栈" << endl;

}

else if (Action[S][m][1] == '4')

//状态10 期待运算符号或者)，但输入是$ ->缺少右括号

{

Value[top] = ')';

State[top] = 16;

cout << "缺少右括号！将)压入栈" << endl;

}

else if (Action[S][m][1] == '5')

//状态1 10 期待+或-号，但输入是\*或/

{

Value[top] = '+';

State[top] = 6;

cout << "缺少+/-符号！将+压入栈" << endl;

}

else if (Action[S][m][1] == '6')

//归约时输入为(或者num

{

ip ++;

cout << "缺少运算符号！跳过" << endl;

}

else

//小数点后缺数字

{

top ++;

Value[top] = 0;

State[top] = 17;

cout << "小数点后缺数字！" << endl;

}

//break;

}

}

return;

}

int main()

{

Init();

LR\_analysis();

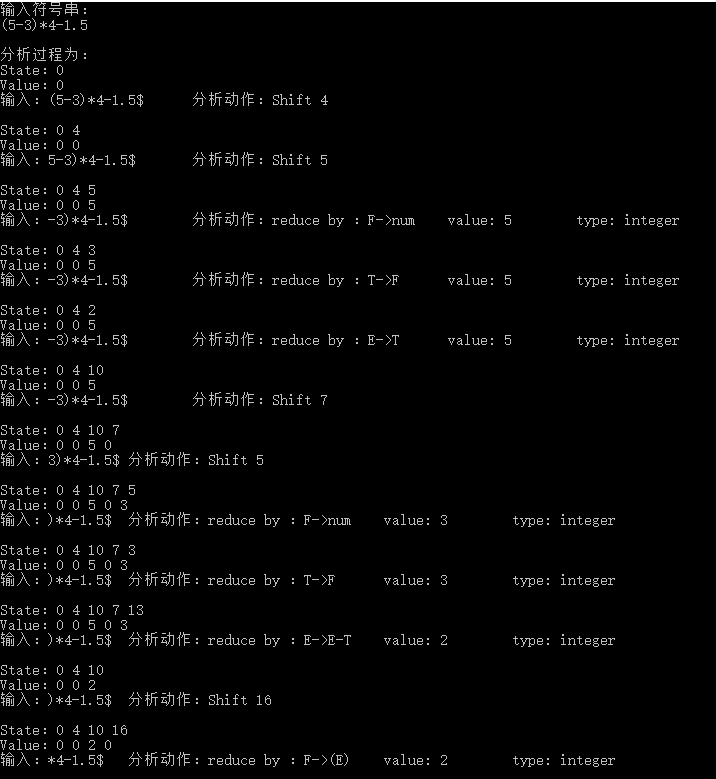
system("pause");

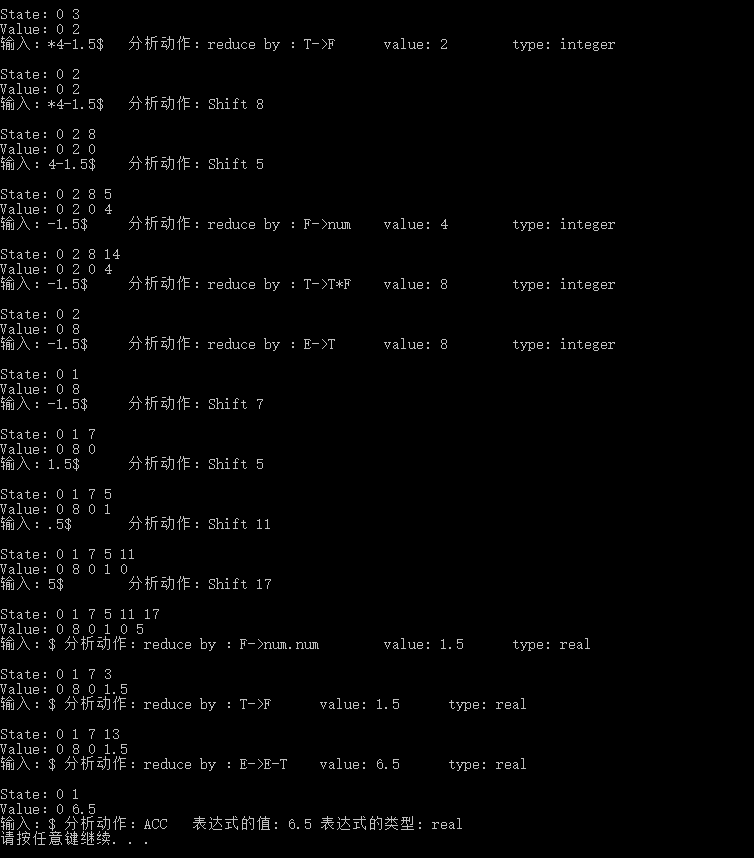
return 0;

}

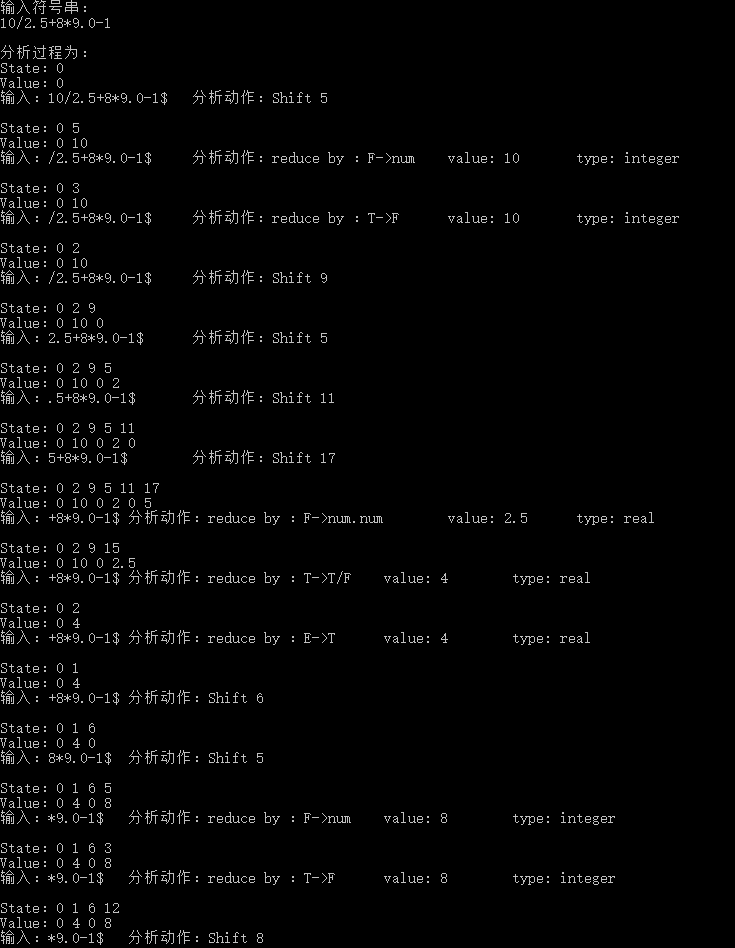
**四、程序测试说明**

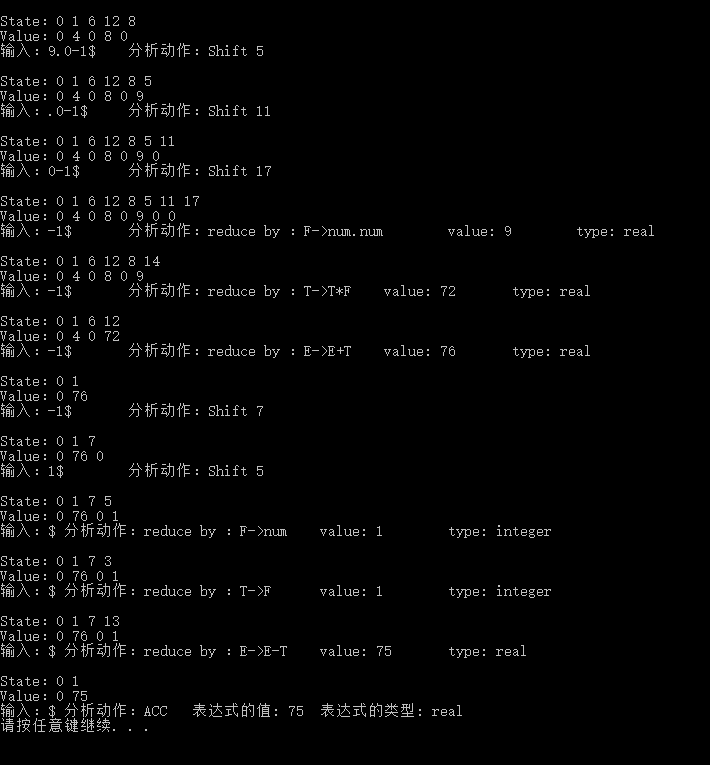
（1）输入为(5-3)\*4-1.5，无错误产生





（2）输入为10/2.5+8\*9.0-1，无错误产生





（3）8\*2.-3+)6/5.，出现三个错误



