**LR语法分析程序实验报告**

**班级：2014211302**

**学号：2014211168**

**姓名：周尧棋**

**一、实验要求**

编写语法分析程序，实现对算术表达式的语法分析。要求所分析的算术表达式由如下文法产生。

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

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

F->(E)|num

在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。

编写语法分析程序实现自顶向上的分析，要求如下。

（1）构造识别该文法所有活前缀的DFA

（2）构造该文法的LR分析表

（3）编程实现算法4.3，构造LR分析程序

**二、程序设计说明**

**1.整体说明**

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getP()获取从键盘输入的文法生成式，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。

**2.构造 LR(0)项目集规范族及识别其所有活前缀的DFA**

首先构造文法的拓广文法G'：

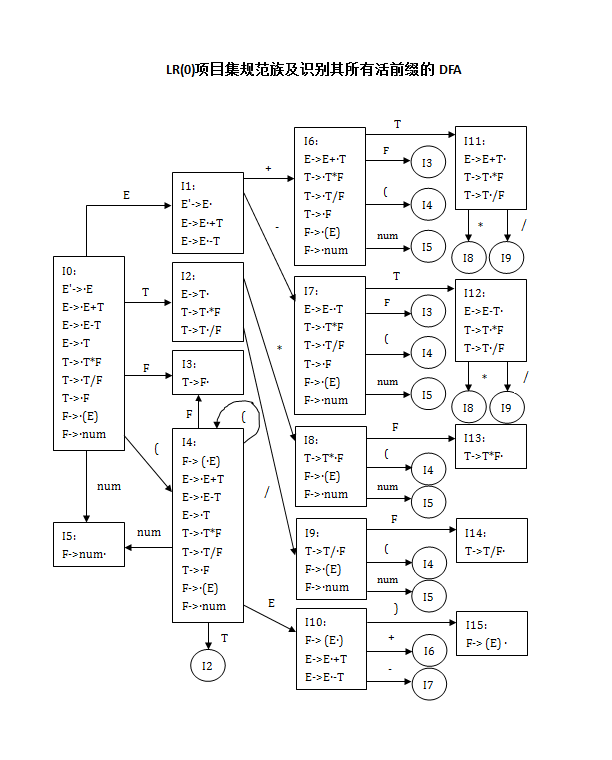
E'->E

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

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

F->(E)|num

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：

****

**3.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。当然要先求出各个非终结符的FOLLOW集，FOLLOW(E') = { $ }，FOLLOW(E) = { +, -, ), $ }，FOLLOW(T) = { +, -, \*, /, ), $ }， FOLLOW(F) = { +, -, \*, /, ), $ }。

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”。

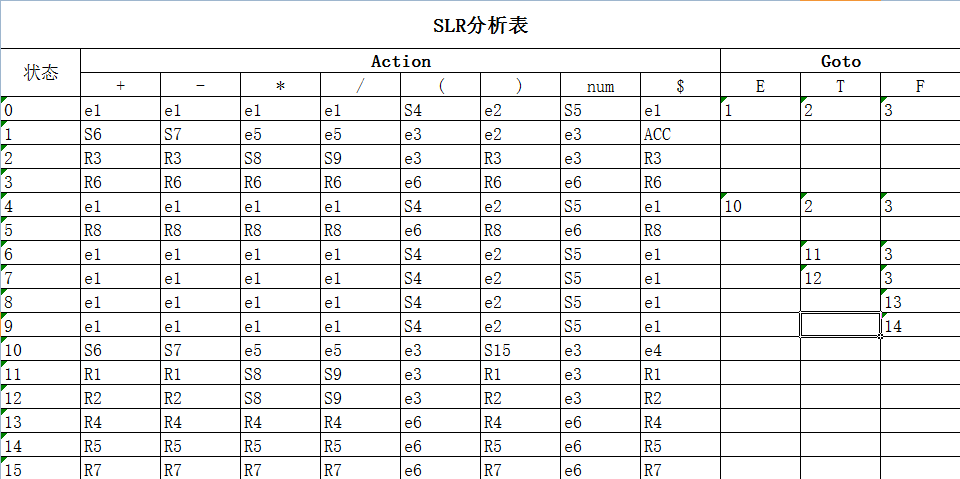
e3：状态1、2、10 、11、12期待运算符号或)，但输入是(或num，打印错误“缺少运算符”。

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”。

最终得到LR分析表，结果如下图

****

**4.LR分析程序**

（1）主要数据结构

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

stack<int> State; //LR分析过程中的状态栈

stack<string> Symble; //LR分析过程中的符号栈

//LR分析表中的action表

string Action[20][20] =

{

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

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

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

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

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

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

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

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

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

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

{"S6","S7","e5","e5","e3","S15","e3","e4"},

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

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

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

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

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

};

//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},

{-1,11,3},

{-1,12,3},

{-1,-1,13},

{-1,-1,14},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

};

（2）主要函数

void getP() //取从键盘输入的文法生成式，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。

void output\_stack(int i) //按栈底->栈顶顺序从左到右打印栈中元素，输入的i值为0打印状态栈State，输入的i值为1打印状态栈Symble。

void LR\_analysis() //分析主程序，按照算法4.3进行操作，加入了错误处理机制。

**三、源程序**

#include<iostream>

#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

stack<int> State;

stack<string> Symble;

//初始化LR分析表中的action表

string Action[20][20] =

{

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

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

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

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

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

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

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

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

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

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

{"S6","S7","e5","e5","e3","S15","e3","e4"},

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

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

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

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

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

};

//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},

{-1,11,3},

{-1,12,3},

{-1,-1,13},

{-1,-1,14},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

{-1,-1,-1},

};

string CharToString(char c)

//将char型转换为string

{

string str;

stringstream stream;

stream << c;

str = stream.str();

return str;

}

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 addToVT(string vt)

//加入VT集

{

if (vt.length() > 0)

{

if (vt == "num")

{

if (finds(VT,vt) < 0)

{

VT.push\_back(vt);

}

}

else

{

for (int j = 0; j < vt.length(); j ++)

{

if (finds(VT,CharToString(vt[j])) < 0)

{

VT.push\_back(CharToString(vt[j]));

}

}

}

}

}

void getP()

//输入产生式

{

string s;

string temp;

string vt;

cout << "用~代替了ε！" << endl;

cout << "输入产生式：（结束输入over）" << endl;

cin >> s;

while (s != "over")

{

temp.clear();

vt.clear();

if(s.length() > 1)

{

for (int i = 0; i < s.length(); i ++)

{

if(s[i] >= 'A' && s[i] <= 'Z')

{

if (findVN(VN,s[i]) < 0)

{

VN.push\_back(s[i]);

}

if (i != 0)

{

temp += s[i];

addToVT(vt);

vt.clear();

//cout << temp << endl;

}

}

else if ((s[i] == '-' && s[i+1] == '>') || (s[i] == '>' && s[i-1] == '-'))

{

continue;

}

else if (s[i] == '|')

{

addToVT(vt);

vt.clear();

string j = "->";

P.push\_back(s[0] + j + temp);

temp.clear();

}

else

{

vt += s[i];

temp += s[i];

}

if (i == s.length() - 1)

{

addToVT(vt);

vt.clear();

string j = "->";

P.push\_back(s[0] + j + temp);

temp.clear();

}

}

}

cout << "输入产生式：（结束输入over）" << endl;

cin >> s;

}

vector<char>::iterator it;

vector<string>::iterator it1;

cout << "VN:" ;

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

{

cout << \*it << " ";

}

cout << endl;

cout << "VT:" ;

for (it1 = VT.begin(); it1 != VT.end(); it1 ++)

{

cout << \*it1 << " ";

}

cout << endl;

for (it1 = P.begin(); it1 != P.end(); it1 ++)

{

cout << \*it1 << endl;

}

cout << endl;

}

void output\_stack(int i)

//打印栈中的内容

{

stack<int> R1;

stack<string> R2;

int carry1;

string carry2;

if (i == 0) //打印State栈

{

while(!State.empty())

{

carry1 = State.top();

State.pop();

R1.push(carry1);

}

while(!R1.empty())

{

carry1 = R1.top();

cout << carry1 << " ";

R1.pop();

State.push(carry1);

}

}

else //打印Symble栈

{

while(!Symble.empty())

{

carry2 = Symble.top();

Symble.pop();

R2.push(carry2);

}

while(!R2.empty())

{

carry2 = R2.top();

cout << carry2 << " ";

R2.pop();

Symble.push(carry2);

}

}

}

void LR\_analysis()

{

string input;

cout << "输入符号串：" << endl;

cin >> input;

cout << endl;

input += "$";

State.push(0); //将状态0压入State栈中

int i = 0;

int j;

int S;

string temp,carry;

VT.push\_back("$");

cout << "分析过程为：" << endl;

while(1)

{

S = State.top(); //S为当前State栈栈顶元素

j = i;

while (finds(VT,temp) < 0) //取得剩余输入串的第一个符号

{

temp += input[j++];

}

cout << "State：" ;

output\_stack(0);

cout << endl;

cout << "Symble：" ;

output\_stack(1);

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

if (Action[S][finds(VT,temp)][0] == 'S')

//移进操作

{

State.push(Action[S][finds(VT,temp)][1] - 48); //将状态号转为int型

Symble.push(temp);

cout << "Shift " << Action[S][finds(VT,temp)][1] << endl << endl;

temp.clear();

i = j; //指向剩余输入串下一个符号

}

else if (Action[S][finds(VT,temp)][0] == 'R')

//归约操作

{

int r = Action[S][finds(VT,temp)][1] - 48;

int cnt = 0;

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 ++)

{

State.pop();

Symble.pop();

}

//\_S为栈顶当前状态

int \_S = State.top();

//将A压入Symble栈中

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

Symble.push(A);

//将goto[\_S,A]压入State栈中

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

State.push(Goto[\_S][vn]);

//输出A->b

cout << "reduce by ：" << P[r-1] << endl << endl ;

}

else if (Action[S][finds(VT,temp)][0] == 'A')

//接收

{

cout << "ACC" << endl;

break;

}

else if (Action[S][finds(VT,temp)][0] == 'e')

//错误处理

{

cout << "error! ";

//break;

if (Action[S][finds(VT,temp)][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

{

Symble.push("num");

State.push(5);

cout << "缺少运算对象！将num压入栈" << endl;

}

else if (Action[S][finds(VT,temp)][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

{

i ++;

cout << "括号不匹配！删掉“）”" << endl;

}

else if (Action[S][finds(VT,temp)][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

{

Symble.push("+");

State.push(6);

cout << "缺少运算符号！将+压入栈" << endl;

}

else if (Action[S][finds(VT,temp)][1] == '4')

//状态10 期待运算符号或者)，但输入是$ ->缺少右括号

{

Symble.push(")");

State.push(15);

cout << "缺少右括号！将)压入栈" << endl;

}

else if (Action[S][finds(VT,temp)][1] == '5')

//状态1 10 期待+或-号，但输入是\*或/

{

Symble.push("+");

State.push(6);

cout << "缺少+/-符号！将+压入栈" << endl;

}

else

//规约时输入为(或者num

{

if (temp == "(")

{

i ++;

}

else

{

i = i + 3;

}

cout << "缺少运算符号！跳过" << endl;

}

}

temp.clear();

}

cout << endl;

cout << "finished!" << endl;

}

int main()

{

getP();

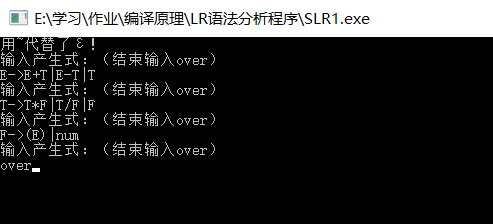
LR\_analysis();

system("pause");

return 0;

}

**四、程序测试说明**

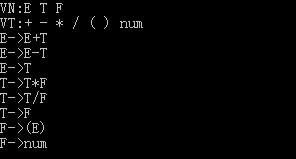


首先输入算术表达式文法（键盘输入时ε用~代替）

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

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

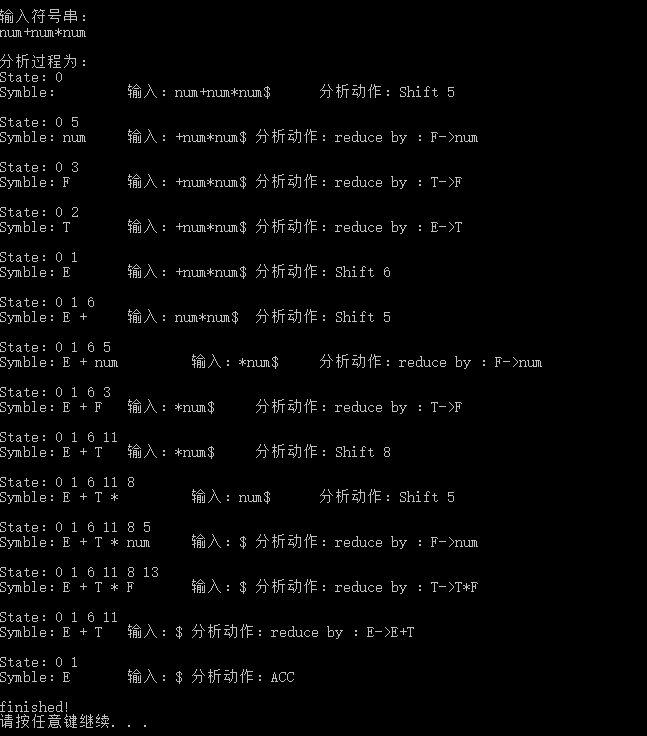
F->(E)|num



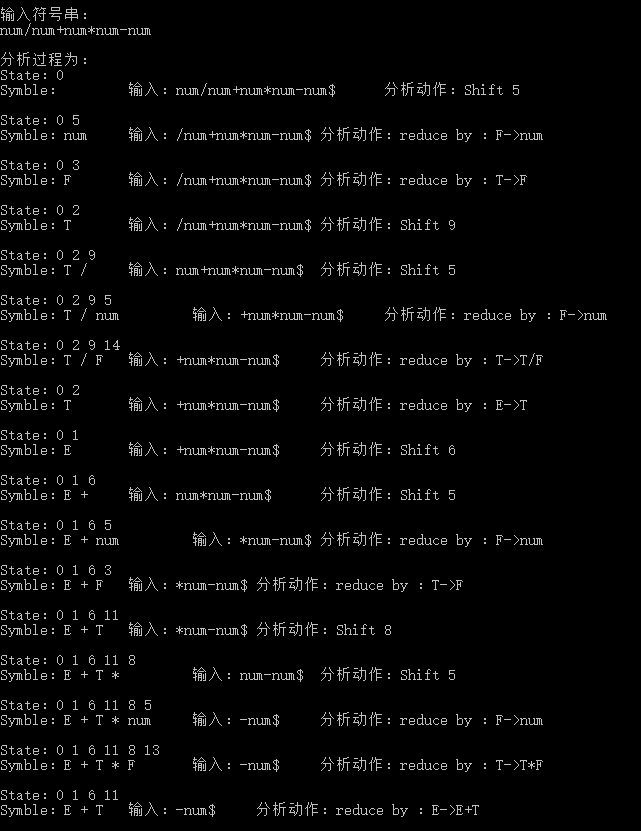
getP()函数根据输入的产生式得到VN和VT集，并将产生式存入容器P中。

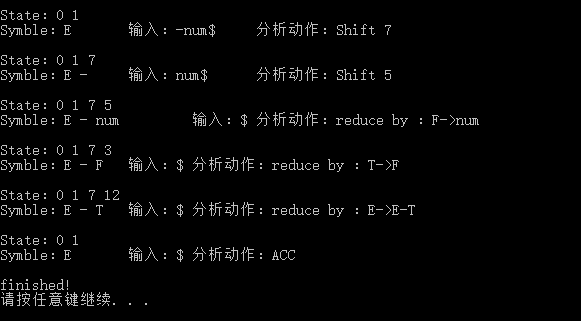
最后根据LR\_analysis()函数得到分析过程

当输入符号串为num+num\*num时，分析动作如下图，发现分析过程中没有发现错误，最后输出接受态ACC。

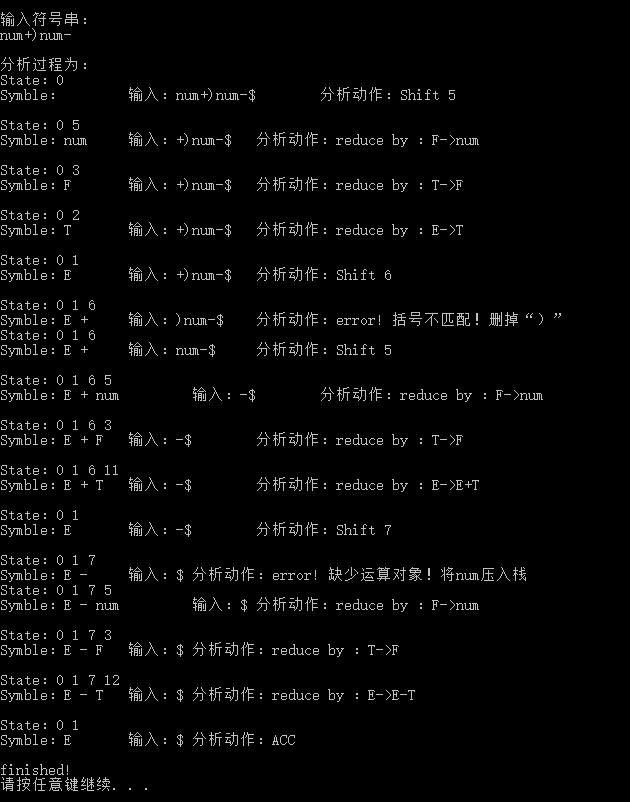


当输入符号串为num/num+num\*num-num时，依然没有错误发生，最后输出接受态ACC，分析过程如下图：





当输入字符串为num+)num-时，在分析过程中会提示错误，但通过相应的错误处理机制，可以使程序继续进行下去。如当栈中为E+，而输入符号为)，提示错误“括号不匹配”，然后跳过括号继续向下分析；还有在栈中元素为E-，而输入符号为$时，提示错误“缺少运算对象”，将一个假想的num入符号栈，然后状态栈转到5，继续进行分析，最后为接受态ACC。



**五、心得与体会**

此次LR分析程序的编写中关键的一步是构造 LR(0)项目集规范族及识别其所有活前缀的DFA，画图需要很细心，值得改进的地方是，如果能够通过编程来得到DFA效率会不会更高，这是需要继续改进的地方。此外，LR分析表的构造也许也可以通过编程实现，这也是需要改进的地方。LR分析过程按照LR分析表来写还是不难的，对于不同情况的错误处理需要仔细思考，什么时候跳过输入的符号，什么时候压入一个假想的符号，以便于分析程序的继续进行。