编译器专题实验报告

**实验五：语义分析（独立模式）**

**一、实验目的**

构建语法制导的语义分析程序能在语法分析的同时生成符号表和中间语言代码，并输出结果到文件中。

实验实现以下功能：

SLR(1)制导的语义分析框架实现；

中间语言代码形式，三元式或四元式，或逆波兰表达式。

**二、实验一（必做）**

**1.实验要求**

根据之前的代码实现的SLR分析表，设计语法制导翻译过程，设计中间代码四元式或者三元式或者逆波兰式分析过程。

本报告利用上次实验的SLR分析程序实现生成逆波兰式的过程。比如，输入a+b，输出ab+。

**2. 实验过程**

**（1）产生式文法**

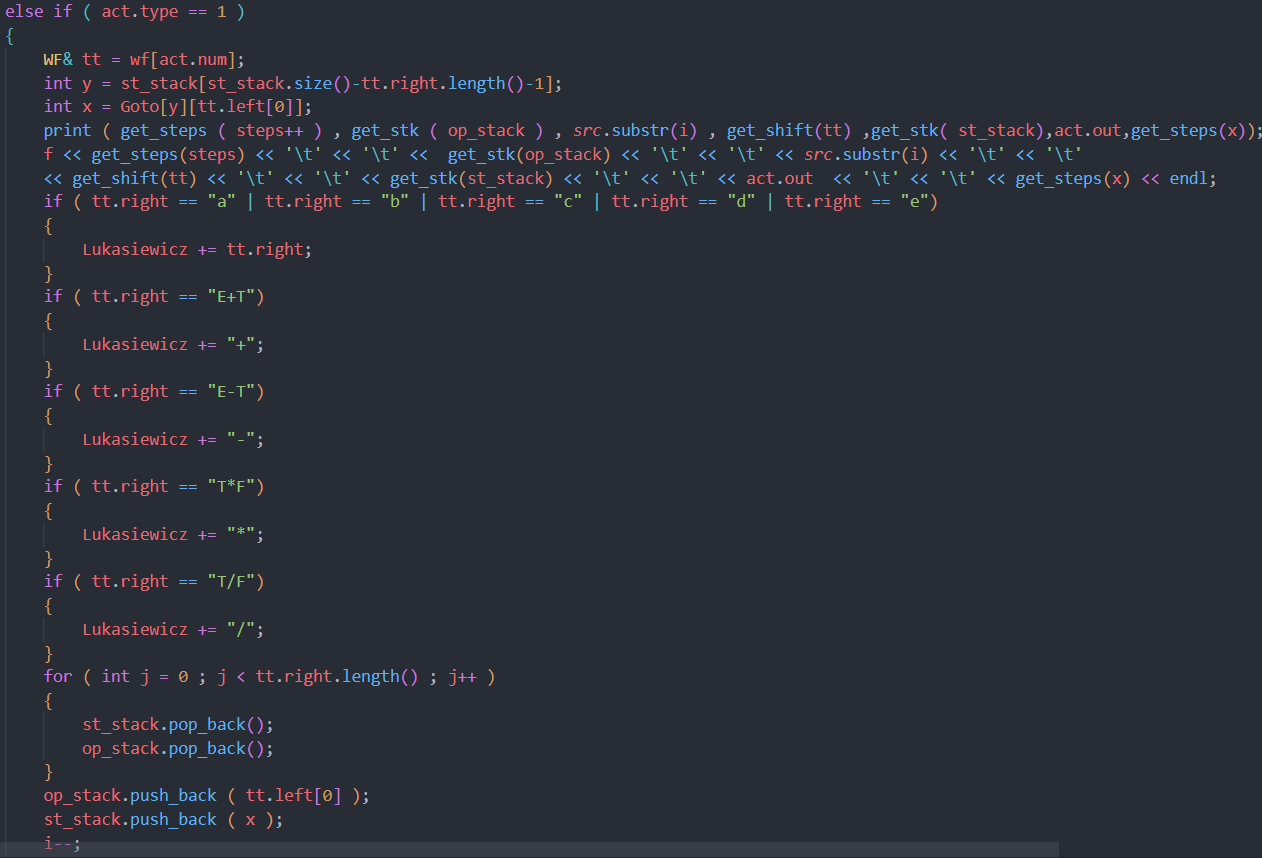
本次实验使用abcde作为操作数，可以处理加减乘除括号运算，使用的产生式如下所示：

S->E E->E+T E->E-T E->T T->T\*F T->T/F T->F F->(E)

F->a F->b F->c F->d F->e

**（2）程序设计**

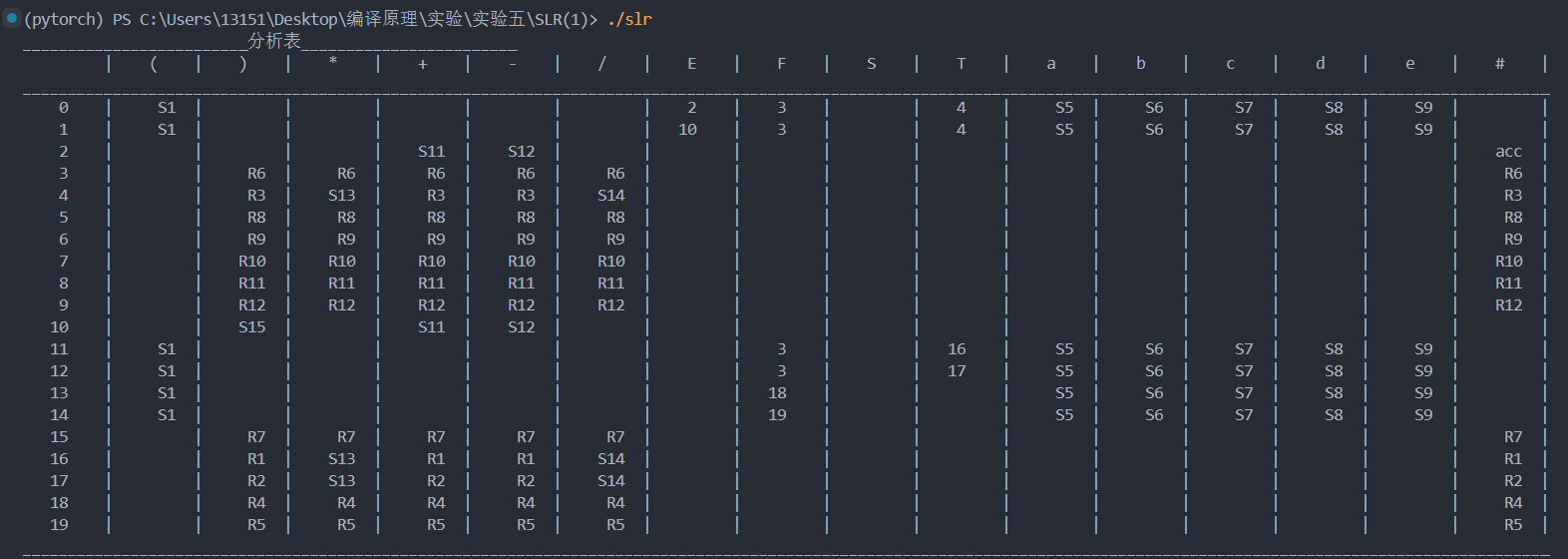
将上述产生式存在grammar.txt中，使用上次实验的SLR(1)分析代码读取文件中的内容并产生SLR(1)分析表，随后读入字符串，使用该分析表进行语法制导的语义分析，与上次实验不同的是，当使用产生式规约时，需要进行对应的操作，当形如使用X->Y op Z的产生式规约时，需要输出op，另外，当使用形如F->id 的产生式规约时需要输出id。所以此次实验需要在上次实验的基础上进行分析过程函数的修改，修改如下所示。



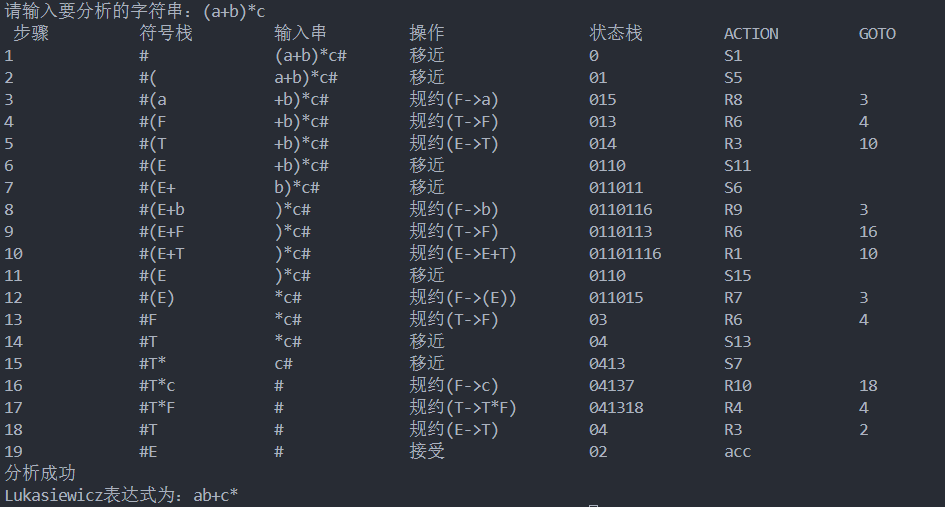
简单来说，就是在上次实验基础上，在SLR进行规约(act.type=1)时进行操作数和运算符的识别操作，当使用X->Y op Z或F->id 进行规约时，将op或id存入一个字符串中，最终进行字符串的输出，即为逆波兰式。

**（3）实验验证**

编译并运行上述程序，首先会根据读入的文法生成SLR分析表，如下所示：



随后程序将会要求读入需要转换的字符串，这里我们以(a+b)\*c为例，输入字符串后，程序会执行处理过程并输出逆波兰表达式ab+c\*。

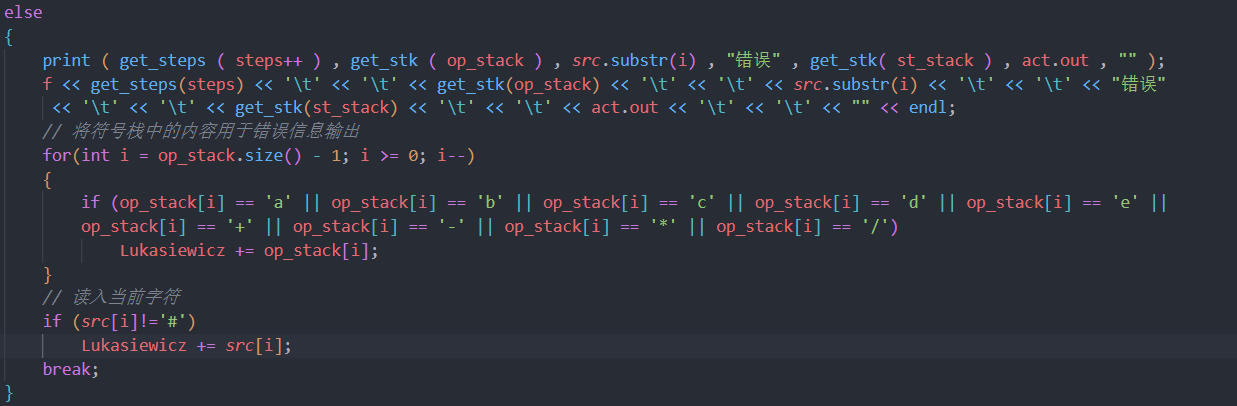


**三、实验二（选做）**

**1. 错误处理**

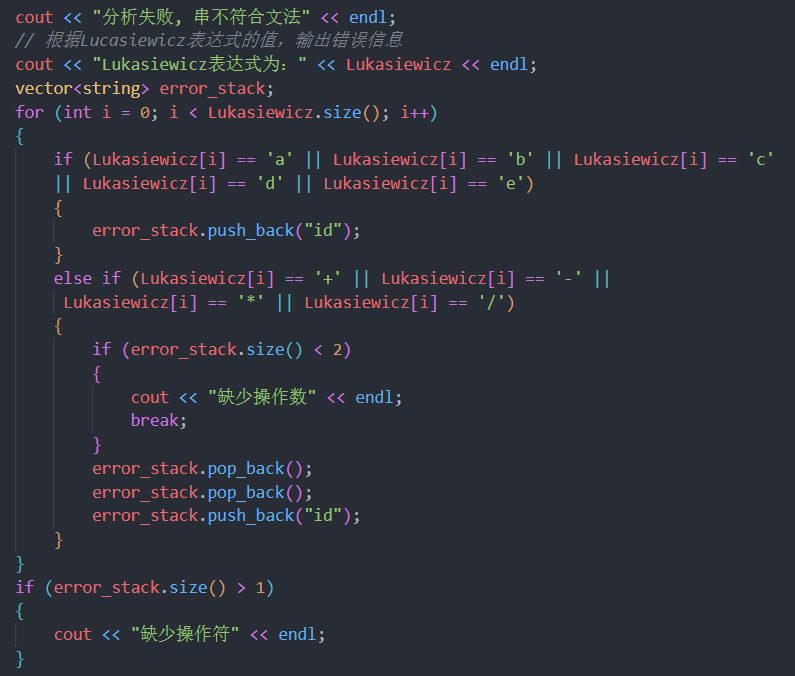
**（1）文法无法处理的情况**

对于SLR分析表中空缺的内容（不进行移进或者规约），需要进行报错处理并终止分析。



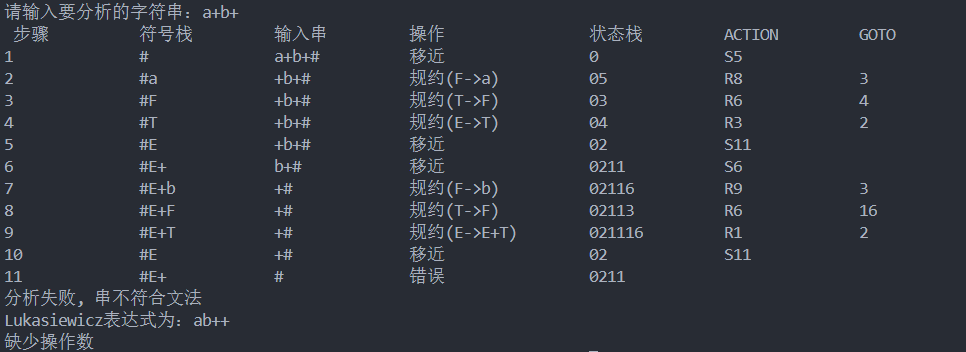
**（2）判断缺少操作数或者操作符**

发生错误还有可能是因为输入的内容缺少操作数或者操作符，本程序可以综合读入的字符串进行逆波兰式是否缺少操作符的判断，代码如下所示。

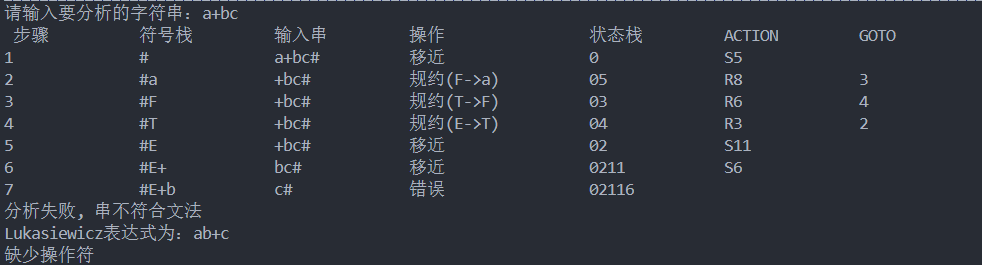


**（3）实验验证**

当输入内容为a+b+时，输入内容缺少操作数，程序会发现当前文法无法接收输入串，并返回缺少操作数的错误。



当输入内容为ab+c时，输入内容缺少操作符，程序会发现当前文法无法接收输入串，并返回缺少操作符的错误。



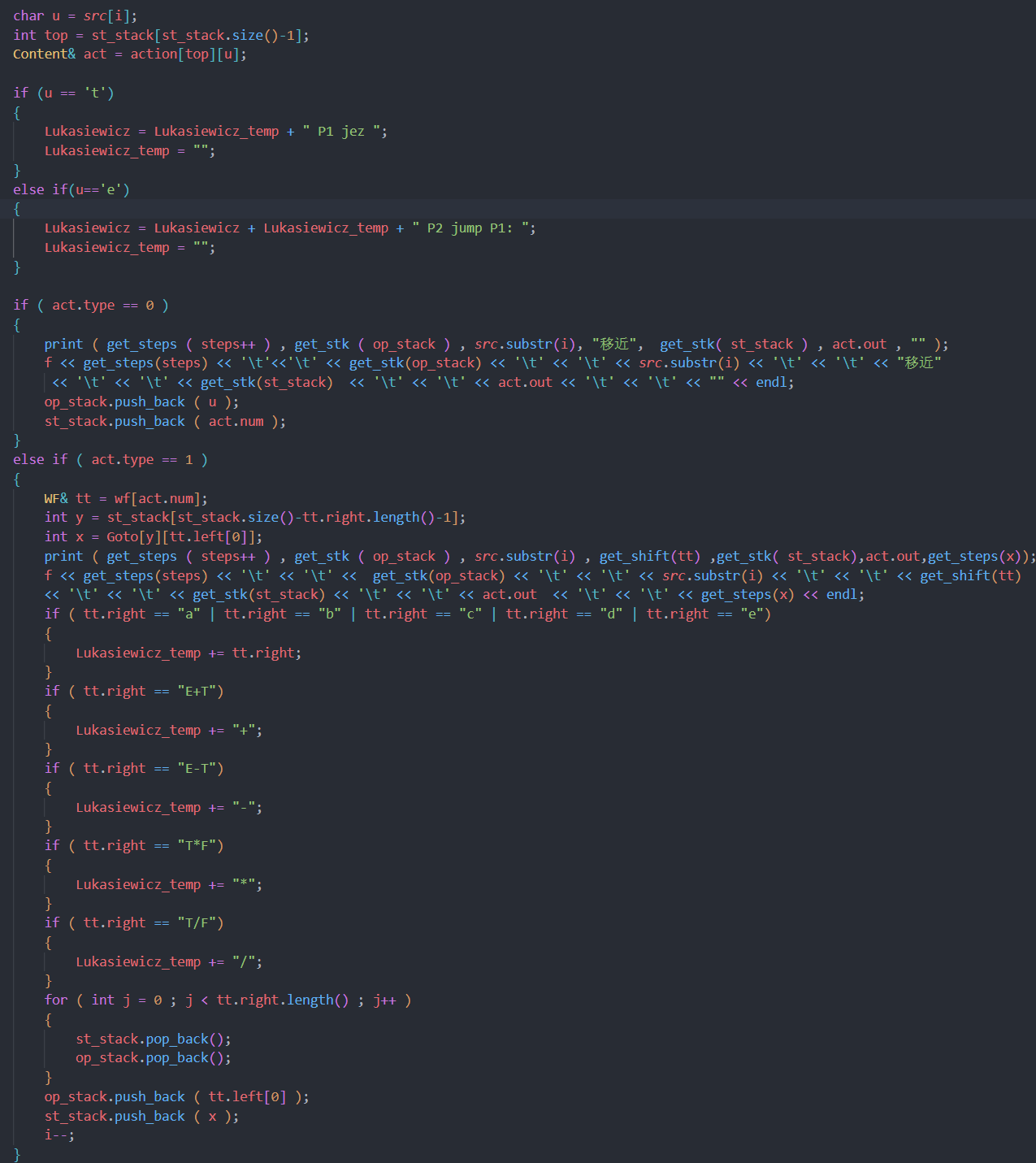
**2. if语句的处理**

**（1）产生式规则编写**

本次实验仅考虑一条if then else 语句的逆波兰式的生成，需要添加产生式：S->H H->i(E)t(E)e(E)，其中i表示if,t表示then,e表示else。当输出i(x)t(y)e(z)，即if x then y else z时，需要输出x P1 jez y P2 jump P1: z P2:...

**（2）程序编写**

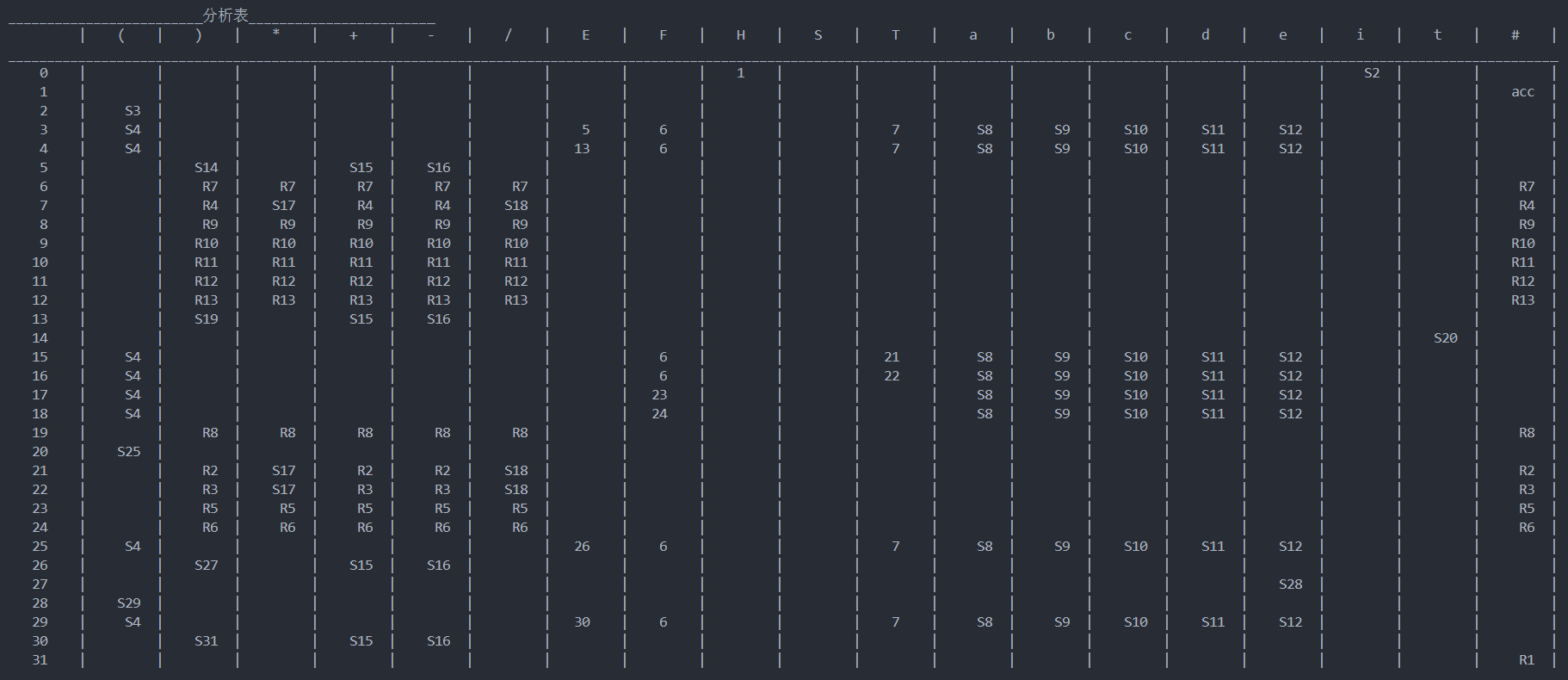
和必做内容类似，只需要进行分析程序的改写即可。其主要修改内容如下。



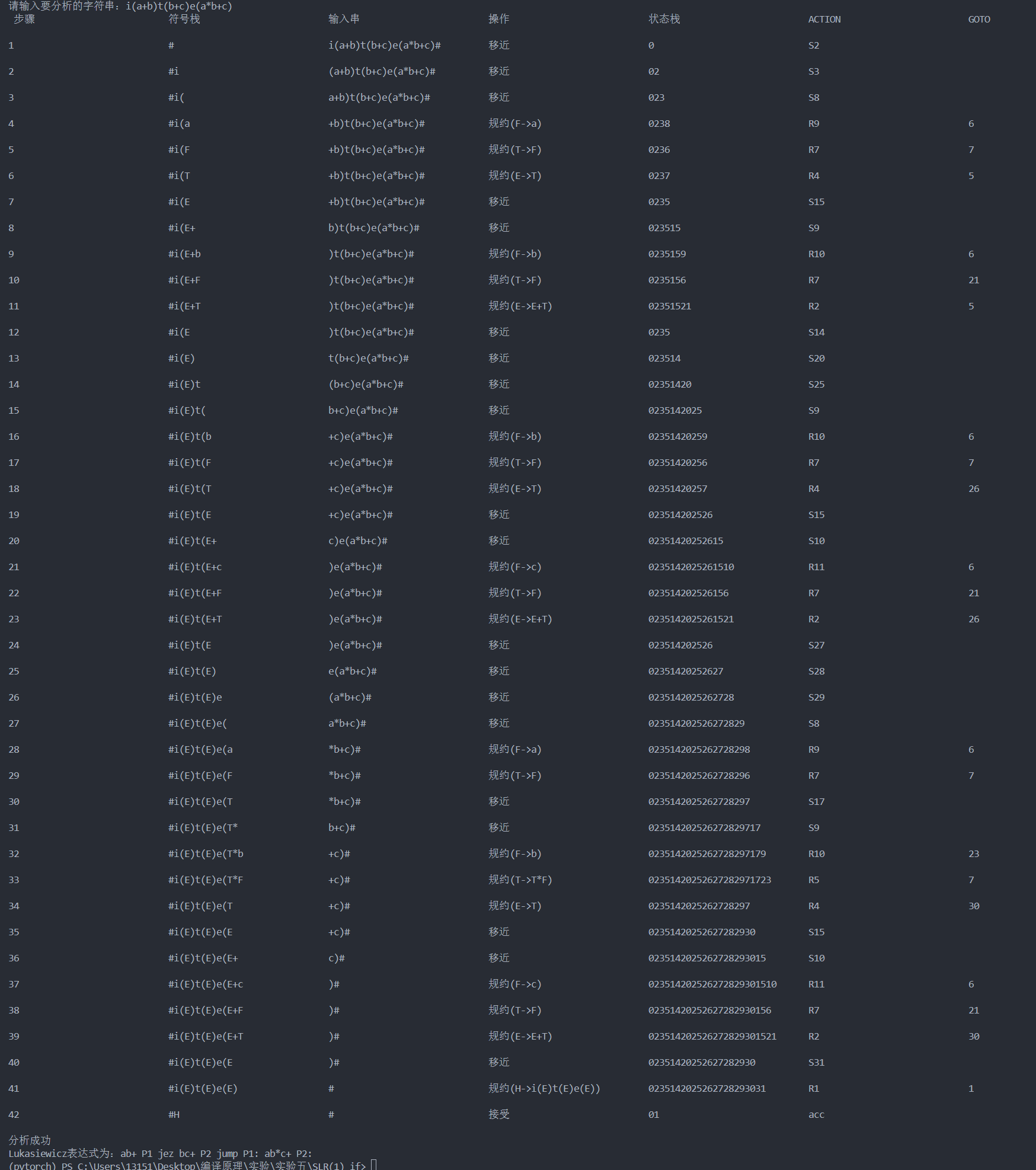
也就是对于一条if语句，需要记录其中可能产生的三段逆波兰式，并使用if语句的逆波兰式进行拼接。

**（3）实验验证**

由于if的产生式和之前的产生式不一样，所以其产生的分析表也不一样，if的分析表如下。



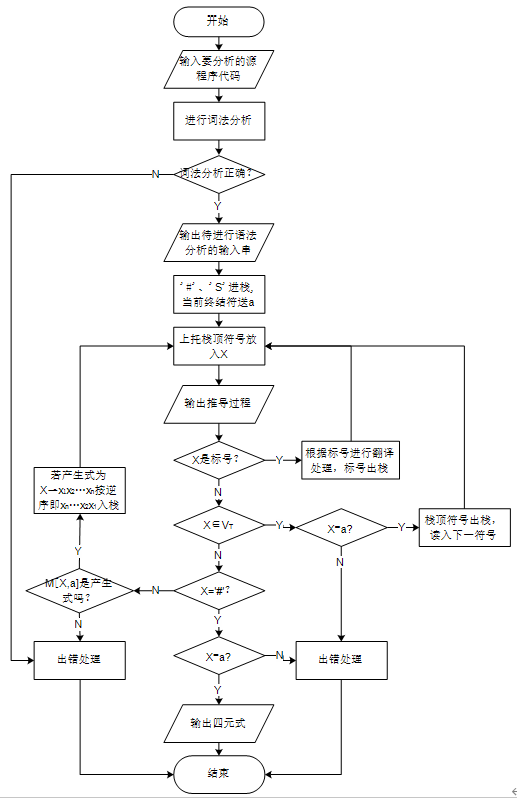
输入i(a+b)t(b+c)e(a\*b+c)会生成规约过程和逆波兰表达式如下。



会生成表达式：ab+ P1 jez bc+ P2 jump P1: ab\*c+ P2:...

**3. while语句的处理**

这里使用LL(1)法进行while语句的语义分析，输出四元式，参考https://github.com/ZhangChaojieZ/whut-cs-semantic\_analysis。程序设计的框架如下所示。



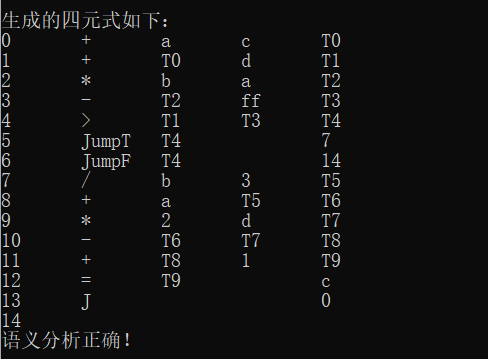
使用程序进行如下代码的分析

while(a + c + d > b \* a - ff){

c = a + b / 3 - 2 \* d + 1;

}

可以得到语义分析结果



**四、遇到的问题**

《编译原理》控制流语句 if 和 while 语句的翻译https://www.cnblogs.com/xpwi/p/11072234.html

**五、附件：代码**

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <cstring>

#include <cctype>

#include <vector>

#include <string>

#include <queue>

#include <map>

#include <set>

#include <sstream>

#include<fstream>

#define MAX 507

#define DEBUG

//#ifdef \_DEBUG的意思是如果预定义了名字\_DEBUG，则函数执行该语句里的代码, 并且抛弃下面的代码

//否则抛弃上面的代码而执行#else里的，

//#endif表示该预处理结束。

using namespace std;

//文法类

class WF

{

public:

string left,right;//产生式左边和右边

int back;

int id;//项目集序号

WF ( char s1[] , char s2[] , int x , int y )

{

left = s1;

right = s2;

back = x;

id = y;

}

WF ( const string& s1 , const string& s2 , int x , int y )

{

left = s1;

right = s2;

back = x;

id = y;

}

//重载<运算符

bool operator < ( const WF& a ) const

{

if ( left == a.left )

return right < a.right;

return left < a.left;

}

//重载==

bool operator == ( const WF& a ) const

{

return ( left == a.left )&& ( right == a.right );

}

void print ( )

{

// printf ( "%s->%s\n" , left.c\_str() , right.c\_str() );

}

};

//闭包类

class Closure

{

public:

vector<WF> element;

void print ( string str )

{

/\* printf ( "%-15s%-15s\n" , "" , str.c\_str());

for ( int i = 0 ; i < element.size() ; i++ )

element[i].print();\*/

}

//重载==

bool operator == ( const Closure& a ) const

{

if ( a.element.size() != element.size() ) return false;

for ( int i = 0 ; i < a.element.size() ; i++ )

if ( element[i] == a.element[i] ) continue;

else return false;

return true;

}

};

struct Content

{

int type;

int num;

string out;

Content(){ type = -1; }

Content ( int a , int b )

:type(a),num(b){}

};

vector<WF> wf;

map<string,vector<int> > dic;

map<string,vector<int> > VN\_set;

map<string,bool> vis;

// 文件输入流

ifstream fin;

//定义文法的开始字符

char start;

void start\_zm()

{

char start\_s;

// cout << "请输入文法开始的字符：" << endl;

// cin >> start\_s;

// 从文件中读入

fin >> start\_s;

// cout<<"文法开始字符为："<<start\_s<<endl;

start = start\_s;

}

//string start = "S";

vector<Closure> collection;

vector<WF> items;

char CH = '.';

int go[MAX][MAX];

int to[MAX];

vector<char> V;

bool used[MAX];

Content action[MAX][MAX];

int Goto[MAX][MAX];

map<string,set<char> > first;

map<string,set<char> > follow;

void make\_item ( )

{

memset ( to , -1 , sizeof ( -1 ) );

for ( int i = 0 ; i < wf.size() ; i++ )

VN\_set[wf[i].left].push\_back ( i );

for ( int i = 0 ; i < wf.size() ; i++ )

for ( int j = 0 ; j <= wf[i].right.length() ; j++ )

{

string temp = wf[i].right;

temp.insert ( temp.begin()+j , CH );

dic[wf[i].left].push\_back ( items.size() );

if ( j )

to[items.size()-1] = items.size();

items.push\_back ( WF ( wf[i].left , temp , i , items.size()) );

}

/\*

#ifdef DEBUG

puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_项目集\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

for ( int i = 0 ; i < items.size() ; i++ )

printf("%s->%s id:%d\n", items[i].left.c\_str(), items[i].right.c\_str(), items[i].id);

#endif\*/

}

void dfs ( const string& x )

{

if ( vis[x] ) return;

vis[x] = 1;

vector<int>& id = VN\_set[x];

for ( int i = 0 ; i < id.size() ; i++ )

{

string& left = wf[id[i]].left;

string& right = wf[id[i]].right;

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

if ( isupper(right[j] ) )

{

dfs ( right.substr(j,1) );

set<char>& temp = first[right.substr(j,1)];

set<char>::iterator it = temp.begin();

bool flag = true;

for ( ; it != temp.end() ; it++ )

{

if ( \*it == '~' ) flag = false;

first[left].insert (\*it );

}

if ( flag ) break;

}

else

{

first[left].insert ( right[j] );

break;

}

}

}

void make\_first ( )

{

vis.clear();

map<string,vector<int> >::iterator it2 = dic.begin();

for ( ; it2 != dic.end() ; it2++ )

if ( vis[it2->first] ) continue;

else dfs ( it2->first );

#ifdef DEBUG

map<string,set<char> >::iterator it = first.begin();

for ( ; it != first.end() ; it++ )

{

set<char> & temp = it->second;

set<char>::iterator it1 = temp.begin();

bool flag = false;

for ( ; it1 != temp.end() ; it1++ )

{

if ( flag )

flag = true;

}

}

#endif

// 输出first集

/\*cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_FIRST集\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

for (auto it = first.begin(); it != first.end(); ++it)

{

cout << "FIRST(" << it->first << ")={";

for (auto it1 = it->second.begin(); it1 != it->second.end(); ++it1)

{

cout << \*it1 << " ";

}

cout << "}" << endl;

}\*/

}

void append ( const string& str1 , const string& str2 )

{

set<char>& from = follow[str1];

set<char>& to = follow[str2];

set<char>::iterator it = from.begin();

for ( ; it != from.end() ; it++ )

to.insert ( \*it );

}

bool \_check ( const vector<int>& id, const string str )

{

for ( int i = 0 ; i < id.size() ; i++ )

{

int x = id[i];

if ( wf[x].right == str ) return true;

}

return false;

}

void make\_follow ( )

{

while ( true )

{

bool goon = false;

map<string,vector<int> >::iterator it2 = VN\_set.begin();

for ( ; it2 != VN\_set.end() ; it2++ )

{

vector<int>& id = it2->second;

for ( int i = 0 ; i < id.size() ; i++ )

{

bool flag = true;

WF& tt = wf[id[i]];

string& left = tt.left;

const string& right = tt.right;

for ( int j = right.length()-1 ; j >= 0 ; j-- )

if ( isupper( right[j] ) )

{

if ( flag )

{

int tx = follow[right.substr(j,1)].size();

append( left , right.substr(j,1) );

int tx1 = follow[right.substr(j,1)].size();

if ( tx1 > tx ) goon = true;

if ( \_check ( id , "~" ) )

flag = false;

}

for ( int k = j+1 ; k < right.length() ; k++ )

if ( isupper(right[k] ) )

{

string idd = right.substr(k,1);

set<char>& from = first[idd];

set<char>& to = follow[right.substr(j,1)];

set<char>::iterator it1 = from.begin();

int tx = follow[right.substr(j,1)].size();

for ( ; it1 != from.end() ; it1++ )

if ( \*it1 != '~' )

to.insert ( \*it1 );

int tx1 = follow[right.substr(j,1)].size();

if ( tx1 > tx ) goon = true;

if ( \_check ( id , "~" ) )

break;

}

else

{

int tx = follow[right.substr(j,1)].size();

follow[right.substr(j,1)].insert ( right[k] );

int tx1 = follow[right.substr(j,1)].size();

if ( tx1 > tx ) goon = true;

break;

}

}

else flag = false;

}

}

if ( !goon ) break;

}

#ifdef DEBUG

map<string,set<char> >::iterator it = follow.begin();

for ( ; it != follow.end() ; it++ )

{

set<char> & temp = it->second;

temp.insert ( '#' );

set<char>::iterator it1 = temp.begin();

bool flag = false;

for ( ; it1 != temp.end() ; it1++ )

{

flag = true;

}

}

#endif

// 输出follow集

/\*cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_FOLLOW集\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

for (auto it = follow.begin(); it != follow.end(); ++it)

{

cout << "FOLLOW(" << it->first << ")={";

for (auto it1 = it->second.begin(); it1 != it->second.end(); ++it1)

{

cout << \*it1 << " ";

}

cout << "}" << endl;

}\*/

}

void make\_set ( )

{

bool has[MAX];

for ( int i = 0 ; i < items.size() ; i++ )

//if ( items[i].left[0] == 'S' && items[i].right[0] == CH )

if (items[i].left[0] == start && items[i].right[0] == CH)

{

Closure temp;

string& str = items[i].right;

vector<WF>& element = temp.element;

element.push\_back ( items[i] );

int x = 0;

for ( x = 0 ; x < str.length() ; x++ )

if ( str[x] == CH )

break;

memset ( has , 0 , sizeof ( has ) );

has[i] = 1;

if ( x != str.length()-1 )

{

queue<string> q;

q.push( str.substr(x+1,1) );

while ( !q.empty() )

{

string u = q.front();

q.pop();

vector<int>& id = dic[u];

for( int j = 0 ; j < id.size() ; j++ )

{

int tx = id[j];

if ( items[tx].right[0] == CH )

{

if ( has[tx] ) continue;

has[tx] = 1;

if ( isupper(items[tx].right[1] ) )

q.push ( items[tx].right.substr(1,1));

element.push\_back ( items[tx] );

}

}

}

}

collection.push\_back ( temp );

}

for ( int i = 0 ; i < collection.size() ; i++ )

{

map<int,Closure> temp;

for ( int j = 0 ; j < collection[i].element.size() ; j++ )

{

string str = collection[i].element[j].right;

int x = 0;

for ( ; x < str.length() ; x++ )

if ( str[x] == CH ) break;

if ( x == str.length()-1 )

continue;

int y = str[x+1];

int ii;

//cout << i << "previous: " << str << endl;

str.erase ( str.begin()+x);

str.insert ( str.begin()+x+1 , CH );

//cout << i <<"after: " << str << endl;

WF cmp = WF ( collection[i].element[j].left , str , -1 , -1 );

for ( int k = 0 ; k< items.size() ; k++ )

if ( items[k] == cmp )

{

ii = k;

break;

}

//string& str1 = items[ii].right;

memset ( has , 0 , sizeof ( has ) );

vector<WF>& element = temp[y].element;

element.push\_back ( items[ii] );

has[ii] = 1;

x++;

if ( x != str.length()-1 )

{

queue<string> q;

q.push( str.substr(x+1,1) );

while ( !q.empty() )

{

string u = q.front();

q.pop();

vector<int>& id = dic[u];

for( int j = 0 ; j < id.size() ; j++ )

{

int tx = id[j];

if ( items[tx].right[0] == CH )

{

if ( has[tx] ) continue;

has[tx] = 1;

if ( isupper(items[tx].right[1] ) )

q.push ( items[tx].right.substr(1,1));

element.push\_back ( items[tx] );

}

}

}

}

}

map<int,Closure>::iterator it = temp.begin();

for ( ; it != temp.end() ; it++ )

collection.push\_back ( it->second );

for ( int i = 0 ; i < collection.size() ; i++ )

sort ( collection[i].element.begin() , collection[i].element.end() );

for ( int i = 0 ; i < collection.size() ; i++ )

for ( int j = i+1 ; j < collection.size() ; j++ )

if ( collection[i] == collection[j] )

collection.erase ( collection.begin()+j );

}

/\*

for ( int i = 0 ; i < collection.size() ; i++ )

{

std::cout << "Collection " << i << ":\n";

for (const auto& item : collection[i].element)

{

std::cout << item.left << " -> ";

for (const auto& ch : item.right)

{

std::cout << ch;

}

std::cout << "\n";

}

}\*/

}

void make\_V ( )

{

memset ( used , 0 , sizeof ( used ) );

for ( int i = 0 ; i < wf.size() ; i++ )

{

string& str = wf[i].left;

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

{

if ( used[str[j]] ) continue;

used[str[j]] = 1;

V.push\_back ( str[j] );

}

string& str1 = wf[i].right;

for ( int j = 0 ; j < str1.length() ; j++ )

{

if ( used[str1[j]] ) continue;

used[str1[j]] = 1;

V.push\_back ( str1[j] );

}

}

sort ( V.begin() , V.end() );

V.push\_back ( '#' );

}

void make\_cmp ( vector<WF>& cmp1 , int i , char ch )

{

for ( int j = 0 ; j < collection[i].element.size() ; j++ )

{

string str = collection[i].element[j].right;

int k;

for ( k = 0 ; k < str.length() ; k++ )

if ( str[k] == CH )

break;

if ( k != str.length() - 1 && str[k+1] == ch )

{

str.erase ( str.begin()+k);

str.insert ( str.begin()+k+1 , CH );

cmp1.push\_back ( WF ( collection[i].element[j].left , str , -1 , -1 ) );

}

}

sort ( cmp1.begin() , cmp1.end() );

}

void make\_go ( )

{

memset ( go , -1 , sizeof ( go ) );

int m = collection.size();

for ( int t = 0 ; t < V.size() ; t++ )

{

char ch = V[t];

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

{

vector<WF> cmp1;

make\_cmp ( cmp1 , i , ch );

if ( cmp1.size() == 0 ) continue;

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

{

vector<WF> cmp2;

for ( int k = 0 ; k < collection[j].element.size() ; k++ )

{

string& str = collection[j].element[k].right;

int x;

for ( x = 0 ; x < str.length() ; x++ )

if ( str[x] == CH )

break;

if ( x && str[x-1] == ch )

cmp2.push\_back ( WF( collection[j].element[k].left , str , -1 , -1 ) );

}

sort ( cmp2.begin() , cmp2.end() );

bool flag = true;

if ( cmp2.size() != cmp1.size() ) continue;

for ( int k = 0 ; k < cmp1.size() ; k++ )

if ( cmp1[k] == cmp2[k] ) continue;

else flag = false;

if ( flag )

go[i][ch] = j;

}

}

}

}

void make\_table ( )

{

memset ( Goto , -1 , sizeof ( Goto ) );

//write s to the table

for( int i = 0 ; i < collection.size() ; i++ )

for ( int j = 0 ; j < V.size() ; j++ )

{

char ch = V[j];

int x = go[i][ch];

if ( x == -1 ) continue;

if ( !isupper(ch) )

action[i][ch] = Content ( 0 , x );

else

Goto[i][ch] = x;

}

//write r and acc to the table

for ( int i = 0 ; i < collection.size() ; i++ )

for ( int j = 0 ; j < collection[i].element.size() ; j++ )

{

WF& tt = collection[i].element[j];

if ( tt.right[tt.right.length()-1] == CH )

{

//if ( tt.left[0] == 'S' )

if (tt.left[0] == start)

action[i]['#'] = Content ( 2 , -1 );

else

for ( int k = 0 ; k < V.size() ; k++ )

{

int y = V[k];

if ( !follow[tt.left].count( V[k] ) ) continue;

action[i][y] = Content ( 1, tt.back );

}

}

}

#ifdef DEBUG

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_分析表\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

printf ( "%10s%5c%5s" , "|" , V[0] , "|");

for ( int i = 1 ; i < V.size() ; i++ )

printf ( "%5c%5s" , V[i] , "|" );

puts ("");

for ( int i = 0 ; i < (V.size()+1)\*10 ; i++ )

printf ( "\_" );

puts("");

stringstream sin;

for ( int i = 0 ; i < collection.size() ; i++ )

{

printf ( "%5d%5s" , i , "|" );

for ( int j = 0 ; j < V.size() ; j++ )

{

char ch = V[j];

if ( isupper(ch) )

{

if ( Goto[i][ch] == -1 )

printf ( "%10s" , "|" );

else

printf ( "%5d%5s" , Goto[i][ch] , "|" );

}

else

{

sin.clear();

if ( action[i][ch].type == -1 )

printf ( "%10s" , "|" );

else

{

Content& temp = action[i][ch];

if ( temp.type == 0 )

sin << "S";

if ( temp.type == 1 )

sin << "R";

if ( temp.type == 2 )

sin << "acc";

if ( temp.num != -1 )

sin << temp.num;

sin >> temp.out;

printf ( "%7s%3s" , temp.out.c\_str() , "|" );

}

}

}

puts ("");

}

for ( int i = 0 ; i < (V.size()+1)\*10 ; i++ )

printf ( "\_" );

puts("");

#endif

}

void print ( string s1 , string s2 , string s3 , string s4 , string s5 , string s6 , string s7 )

{

printf("%-15s%-15s%-15s%-20s%-15s%-15s%-15s\n", s1.c\_str(), s2.c\_str(), s3.c\_str(), s4.c\_str(), s5.c\_str(),

s6.c\_str(), s7.c\_str());

}

string get\_steps ( int x )

{

stringstream sin;

sin << x;

string ret;

sin >> ret;

return ret;

}

template<class T>

string get\_stk ( vector<T> stk )

{

stringstream sin;

for ( int i = 0 ; i < stk.size() ; i++ )

sin << stk[i];

string ret;

sin >> ret;

return ret;

}

string Lukasiewicz = "";

string get\_shift ( WF& temp )

{

stringstream sin;

sin << "规约(" << temp.left << "->" << temp.right <<")";

string out;

sin >> out;

return out;

}

void analyse ( string src )

{

print ( " 步骤 ","符号栈" ,"输入串","操作","状态栈 " , "ACTION" , "GOTO" );

string filename;

filename = "analyse.txt";

ofstream f(filename,ios::out);

f << " 步骤 " << '\t' << '\t' << "符号栈" << '\t' << '\t' << "输入串"<< '\t' << '\t' << "操作"

<< '\t' << '\t' << "状态栈 " << '\t' << '\t' << "ACTION" << '\t' << '\t' << "GOTO" << endl;

vector<char> op\_stack;

vector<int> st\_stack;

src+= "#";

op\_stack.push\_back ( '#' );

st\_stack.push\_back ( 0 );

int steps= 1;

bool acc\_flag = false;

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

{

char u = src[i];

int top = st\_stack[st\_stack.size()-1];

Content& act = action[top][u];

if ( act.type == 0 )

{

print ( get\_steps ( steps++ ) , get\_stk ( op\_stack ) , src.substr(i), "移近", get\_stk( st\_stack ) , act.out , "" );

f << get\_steps(steps) << '\t'<<'\t' << get\_stk(op\_stack) << '\t' << '\t' << src.substr(i) << '\t' << '\t'

<< "移近" << '\t' << '\t' << get\_stk(st\_stack) << '\t' << '\t' << act.out << '\t' << '\t' << "" << endl;

op\_stack.push\_back ( u );

st\_stack.push\_back ( act.num );

}

else if ( act.type == 1 )

{

WF& tt = wf[act.num];

int y = st\_stack[st\_stack.size()-tt.right.length()-1];

int x = Goto[y][tt.left[0]];

print ( get\_steps ( steps++ ) , get\_stk ( op\_stack ) , src.substr(i) , get\_shift(tt) ,get\_stk( st\_stack),act.out,get\_steps(x));

f << get\_steps(steps) << '\t' << '\t' << get\_stk(op\_stack) << '\t' << '\t' << src.substr(i) << '\t' << '\t'

<< get\_shift(tt) << '\t' << '\t' << get\_stk(st\_stack) << '\t' << '\t' << act.out << '\t' << '\t' << get\_steps(x) << endl;

if ( tt.right == "a" | tt.right == "b" | tt.right == "c" | tt.right == "d" | tt.right == "e")

{

Lukasiewicz += tt.right;

}

if ( tt.right == "E+T")

{

Lukasiewicz += "+";

}

if ( tt.right == "E-T")

{

Lukasiewicz += "-";

}

if ( tt.right == "T\*F")

{

Lukasiewicz += "\*";

}

if ( tt.right == "T/F")

{

Lukasiewicz += "/";

}

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

{

st\_stack.pop\_back();

op\_stack.pop\_back();

}

op\_stack.push\_back ( tt.left[0] );

st\_stack.push\_back ( x );

i--;

}

else if ( act.type == 2 )

{

acc\_flag = true;

print ( get\_steps( steps++ ), get\_stk( op\_stack ) , src.substr(i) , "接受" , get\_stk(st\_stack) , act.out , "" );

f<<get\_steps(steps) << '\t' << '\t' << get\_stk(op\_stack) << '\t' << '\t' << src.substr(i) << '\t' << '\t' << "接受"

<< '\t' << '\t' << get\_stk(st\_stack) << '\t' << '\t' << act.out << '\t' << '\t' << ""<<endl;

}

else

{

print ( get\_steps ( steps++ ) , get\_stk ( op\_stack ) , src.substr(i) , "错误" , get\_stk( st\_stack ) , act.out , "" );

f << get\_steps(steps) << '\t' << '\t' << get\_stk(op\_stack) << '\t' << '\t' << src.substr(i) << '\t' << '\t' << "错误"

<< '\t' << '\t' << get\_stk(st\_stack) << '\t' << '\t' << act.out << '\t' << '\t' << "" << endl;

// 将符号栈中的内容用于错误信息输出

for(int i = op\_stack.size() - 1; i >= 0; i--)

{

if (op\_stack[i] == 'a' || op\_stack[i] == 'b' || op\_stack[i] == 'c' || op\_stack[i] == 'd' || op\_stack[i] == 'e' ||

op\_stack[i] == '+' || op\_stack[i] == '-' || op\_stack[i] == '\*' || op\_stack[i] == '/')

Lukasiewicz += op\_stack[i];

}

// 读入当前字符

if (src[i]!='#')

Lukasiewicz += src[i];

break;

}

}

f.close();

if ( acc\_flag )

{

cout << "分析成功" << endl;

cout << "Lukasiewicz表达式为：" << Lukasiewicz << endl;

}

else

{

cout << "分析失败, 串不符合文法" << endl;

// 根据Lucasiewicz表达式的值，输出错误信息

cout << "Lukasiewicz表达式为：" << Lukasiewicz << endl;

vector<string> error\_stack;

for (int i = 0; i < Lukasiewicz.size(); i++)

{

if (Lukasiewicz[i] == 'a' || Lukasiewicz[i] == 'b' || Lukasiewicz[i] == 'c'

|| Lukasiewicz[i] == 'd' || Lukasiewicz[i] == 'e')

{

error\_stack.push\_back("id");

}

else if (Lukasiewicz[i] == '+' || Lukasiewicz[i] == '-' ||

Lukasiewicz[i] == '\*' || Lukasiewicz[i] == '/')

{

if (error\_stack.size() < 2)

{

cout << "缺少操作数" << endl;

break;

}

error\_stack.pop\_back();

error\_stack.pop\_back();

error\_stack.push\_back("id");

}

}

if (error\_stack.size() > 1)

{

cout << "缺少操作符" << endl;

}

}

}

int main ( )

{

int n;

char s[MAX];

// 打开文件

fin.open("grammar.txt",ios::in);

start\_zm();

// cout << "请输入文法个数和文法：";

while ( fin >> n )

{

// cout << "文法个数为：" << n << endl;

for ( int i = 0 ; i < n ; i++ )

{

//scanf ( "%s" , s );

fin >> s;

int len = strlen(s),j;

for ( j = 0 ; j < len ; j++ )

if ( s[j] == '-' ) break;

s[j] = 0;

wf.push\_back ( WF ( s , s+j+2 ,-1 , -1 ) );

#ifdef DEBUG

wf[wf.size()-1].print();

#endif

}

make\_item();

make\_first();

make\_follow();

make\_set();

make\_V();

make\_go();

make\_table();

string s1;

cout << "请输入要分析的字符串：";

cin >> s1;

analyse ( s1);

}

}

if语句分析：

string Lukasiewicz = "";

string Lukasiewicz\_temp = "";

string get\_shift ( WF& temp )

{

stringstream sin;

sin << "规约(" << temp.left << "->" << temp.right <<")";

string out;

sin >> out;

return out;

}

void analyse ( string src )

{

print ( " 步骤 ","符号栈" ,"输入串","操作","状态栈 " , "ACTION" , "GOTO" );

string filename;

filename = "analyse.txt";

ofstream f(filename,ios::out);

f << " 步骤 " << '\t' << '\t' << "符号栈" << '\t' << '\t' << "输入串"<< '\t' << '\t' << "操作" << '\t'

<< '\t' << "状态栈 " << '\t' << '\t' << "ACTION" << '\t' << '\t' << "GOTO" << endl;

vector<char> op\_stack;

vector<int> st\_stack;

src+= "#";

op\_stack.push\_back ( '#' );

st\_stack.push\_back ( 0 );

int steps= 1;

bool acc\_flag = false;

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

{

char u = src[i];

int top = st\_stack[st\_stack.size()-1];

Content& act = action[top][u];

if (u == 't')

{

Lukasiewicz = Lukasiewicz\_temp + " P1 jez ";

Lukasiewicz\_temp = "";

}

else if(u=='e')

{

Lukasiewicz = Lukasiewicz + Lukasiewicz\_temp + " P2 jump P1: ";

Lukasiewicz\_temp = "";

}

if ( act.type == 0 )

{

print ( get\_steps ( steps++ ) , get\_stk ( op\_stack ) , src.substr(i), "移近", get\_stk( st\_stack ) , act.out , "" );

f << get\_steps(steps) << '\t'<<'\t' << get\_stk(op\_stack) << '\t' << '\t' << src.substr(i) << '\t' << '\t' << "移近"

<< '\t' << '\t' << get\_stk(st\_stack) << '\t' << '\t' << act.out << '\t' << '\t' << "" << endl;

op\_stack.push\_back ( u );

st\_stack.push\_back ( act.num );

}

else if ( act.type == 1 )

{

WF& tt = wf[act.num];

int y = st\_stack[st\_stack.size()-tt.right.length()-1];

int x = Goto[y][tt.left[0]];

print ( get\_steps ( steps++ ) , get\_stk ( op\_stack ) , src.substr(i) , get\_shift(tt) ,get\_stk( st\_stack),act.out,get\_steps(x));

f << get\_steps(steps) << '\t' << '\t' << get\_stk(op\_stack) << '\t' << '\t' << src.substr(i) << '\t' << '\t' << get\_shift(tt)

<< '\t' << '\t' << get\_stk(st\_stack) << '\t' << '\t' << act.out << '\t' << '\t' << get\_steps(x) << endl;

if ( tt.right == "a" | tt.right == "b" | tt.right == "c" | tt.right == "d" | tt.right == "e")

{

Lukasiewicz\_temp += tt.right;

}

if ( tt.right == "E+T")

{

Lukasiewicz\_temp += "+";

}

if ( tt.right == "E-T")

{

Lukasiewicz\_temp += "-";

}

if ( tt.right == "T\*F")

{

Lukasiewicz\_temp += "\*";

}

if ( tt.right == "T/F")

{

Lukasiewicz\_temp += "/";

}

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

{

st\_stack.pop\_back();

op\_stack.pop\_back();

}

op\_stack.push\_back ( tt.left[0] );

st\_stack.push\_back ( x );

i--;

}

else if ( act.type == 2 )

{

acc\_flag = true;

print ( get\_steps( steps++ ), get\_stk( op\_stack ) , src.substr(i) , "接受" , get\_stk(st\_stack) , act.out , "" );

f<<get\_steps(steps) << '\t' << '\t' << get\_stk(op\_stack) << '\t' << '\t' << src.substr(i) << '\t' << '\t' << "接受"

<< '\t' << '\t' << get\_stk(st\_stack) << '\t' << '\t' << act.out << '\t' << '\t' << ""<<endl;

Lukasiewicz += Lukasiewicz\_temp + " P2: ";

}

else

{

print ( get\_steps ( steps++ ) , get\_stk ( op\_stack ) , src.substr(i) , "错误" , get\_stk( st\_stack ) , act.out , "" );

f << get\_steps(steps) << '\t' << '\t' << get\_stk(op\_stack) << '\t' << '\t' << src.substr(i) << '\t' << '\t' << "错误"

<< '\t' << '\t' << get\_stk(st\_stack) << '\t' << '\t' << act.out << '\t' << '\t' << "" << endl;

break;

}

}

f.close();

if ( acc\_flag )

{

cout << "分析成功" << endl;

cout << "Lukasiewicz表达式为：" << Lukasiewicz << endl;

}

else

{

cout << "分析失败, 串不符合文法" << endl;

}

}