

**《编译原理》**

**实验五实验报告**

|  |  |
| --- | --- |
|  |  |
| 学 号： |  |
| 姓 名： |  |
| 学 院： |  |
| 日 期： |  |

**一、 理论传授**

SLR(1)分析法，语法指导的基本概念，生成四元式的理论和方法。

**二、 目标任务**

本次实验实现一个简单的编程语言的语法制导翻译和中间代码生成。该编程语言由一个名为 f 的“函数”构成，函数名固定为 f，函数需要至少 1 个变量作为参数，变量名只由 1 个小写字母组成，例如 a、b、c 等都是可以作为参数的变量名，但是 f 不可以。函数通过参数列表说明所需要的参数，参数列表紧跟着函数名用方括号包裹列出，如果有多于 1 个的参数，中间使用逗号分开。参数列表后紧跟着一个等号，等号右边是作为函数参数的变量构成的计算表达式，表达式可以有+ - \* /四则运算，括号( )，其运算规则和 C 语言的运算规则一致。但是如果要表达取负数，则需要至少 1 层原括号包裹。举个例子：

f[x]=x

f[i]=i+i\*i-i/i

f[x,y,z]=x\*y+z-x/(z+x)

分别是使用该语言写成的三个合法的程序，而

f[x]=y

f[x,y,z]=x++y

f[x,y]=xy

f[A]=A

是不合法的程序，第一个程序里表达式中出现了没有在参数列表中声明的变量 y，而第二个、第三个和第四个程序则是出现了语法错误。

于是可以定义出这样的语法：

𝑉𝑡 = {+, −,∗,/, , , (, ),[,], 𝑓, 𝑋} 其中 X 表示除了 f 之外的 任意小写字母 𝑉𝑛 = {S, A, E, T, F},

G[S]:

S → f[A] = E

A → 请你补充A的定义，使其能够表示该编程语言的参 数列表的内容（1 个或多个参数，如果有多个参数中间用 逗号,分开），并且符合 SLR(1)文法。

E → E + T|E − T|T

T → T ∗ F|T/F|𝐹

F → (E)|X

【设计要求】1.补充上面的文法，如果有需要你可以引入新的非终结符号，只要满足要求即可，在实验报告中写出你完善后的文法。2.构造该文法的 SLR(1)分析表。

3.设计语法制导翻译过程，可以给出每一个产生式对应的语义动作（如果有）。

4.测试例子至少应当测试“目标任务”中的所有测试例，对于合法的程序给出程序执行结果的四元式序列，对于错误的程序应该能识别出错误，最好能给出错误信息。

5.四元式可以参考下面的设计，也可以自行设计。

定义函数：

(DEF\_FUNC, 函数名,,)

定义 1 个参数：

(DEF\_ARG, 参数名,,)

并且，所有的 DEF\_ARG 一定会紧跟在 DEF\_FUNC 后面

加法：

(ADD, 源操作数 1,源操作数 2,目标操作数)

减法：

(SUB, 源操作数 1,源操作数 2,目标操作数)

乘法：

(MUL, 源操作数 1,源操作数 2,目标操作数)

除法：

(DIV, 源操作数 1,源操作数 2,目标操作数)

赋值：

(ASSIGN, 源操作数,,目标操作数)

结束函数定义：

(END\_FUNC,,,)

函数的返回值在可以四元式中用特殊的变量 R 表示,参数之外的临时变量可以用 T1,T2,T3…等表示。 例如对于程序 f[x,y]=x\*(x+y)可以翻译为（这只是一种可 能，也可以翻译为其它的等价的四元式序列）:

(DEF\_FUNC, f,,)

(DEF\_ARG, x,,)

(DEF\_ARG, y,,)

(ADD,x,y,T1)

(MUL,x,T1,T2)

(ASSIGN,T2,,R)

(END\_FUNC,,,)

本次实验不要求对中间代码（四元式）优化，有兴趣的同学 可以尝试一下。（例如在上面的例子中，MUL x T1 的结果可以直接给 R，而不需要 T2 中转）

**三、 文法补充**

G[S]:

S → f[A] = E

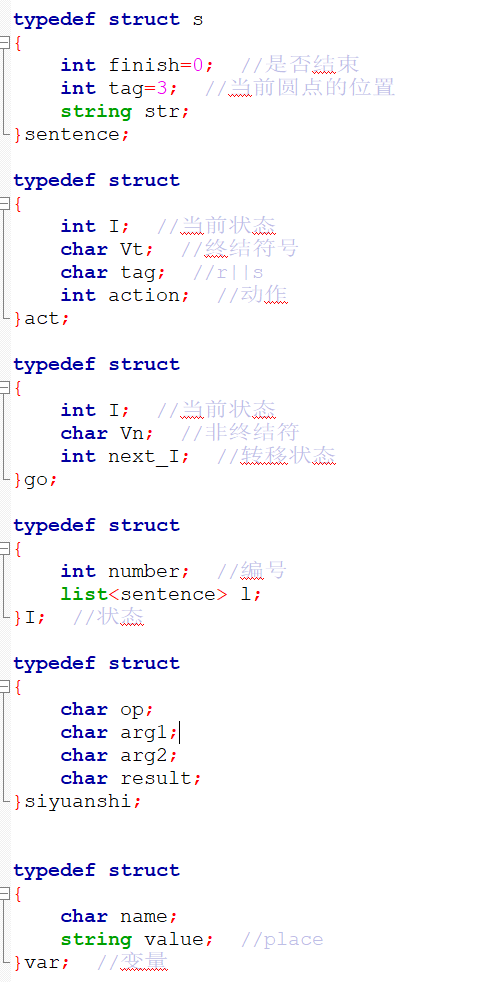
A →A|I,A

E → E + T|E − T|T

T → T ∗ F|T/F|𝐹

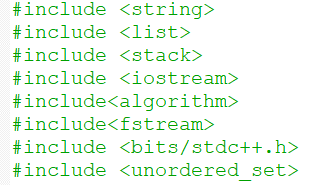
F → (E)|X

1. **数据结构描述**

****

|  |  |
| --- | --- |
| 变量及类型 | 用途 |
| sentence | 用于存放输入的句子，并判断是否匹配 |
| act | 用于存放状态与动作 |
| go | 用于存放当前状态，非终结符和转移状态 |
| I | 用于存放算出的闭包I |
| Siyuanshi | 用于存放和输出四元式 |
| Var | 用于存放变量 |
| list<I> DFA; | 用于存放状态集 |
| char input[100]; | 用于读取词法分析的输入 |
| list<char> Vn; | 用于存放非终结符号集 |
| list<char> Vt; | 用于存放终结符号集 |
| list<go> GOTO; | GO[i][j] |
| list<act> ACTION; | ACTION[i][j] |
| list<char>\* Follow; | 用于存放FOLLOW集 |
| list<char>\* First; | 用于存放FIRST集 |

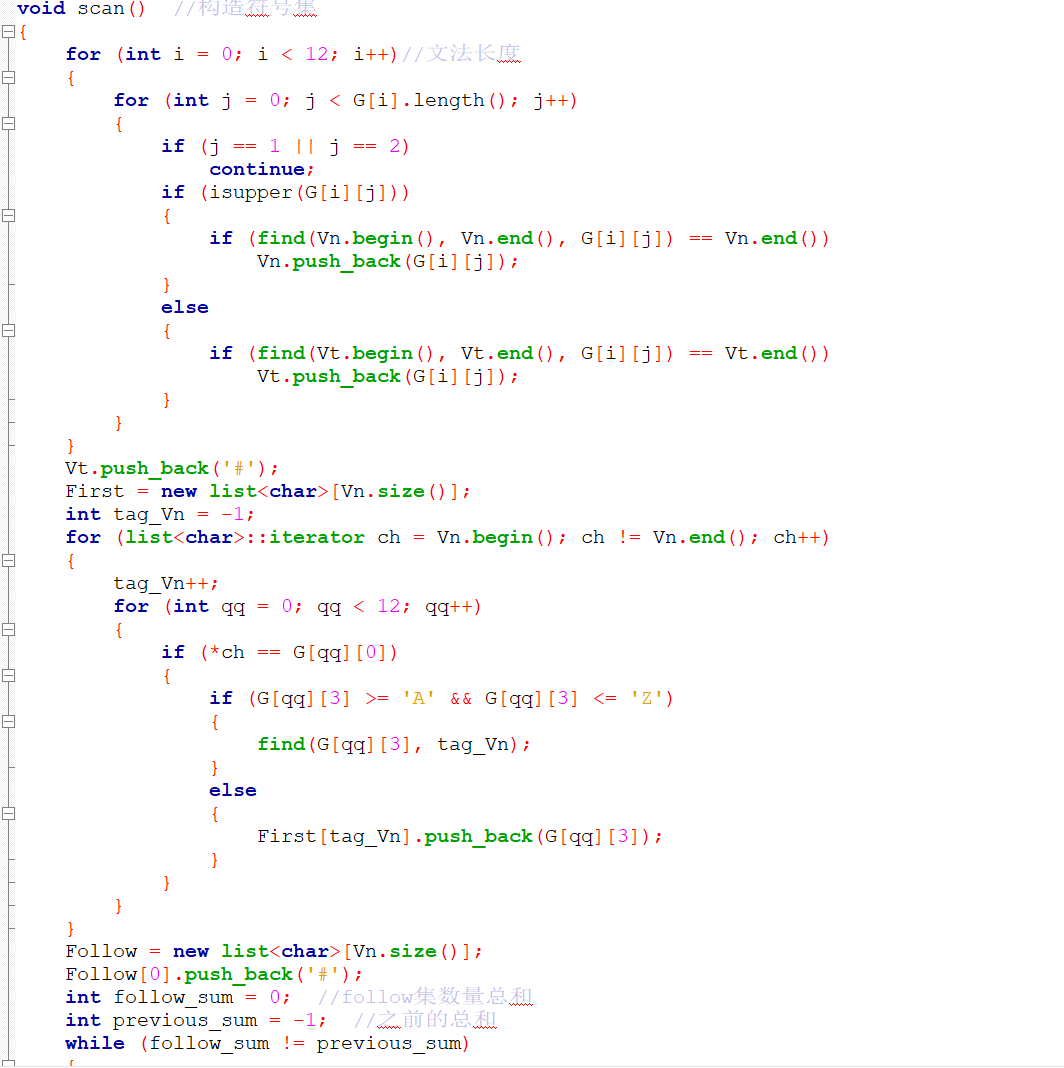
1. **程序结构描述**
2. 设计方法
3. 宏定义

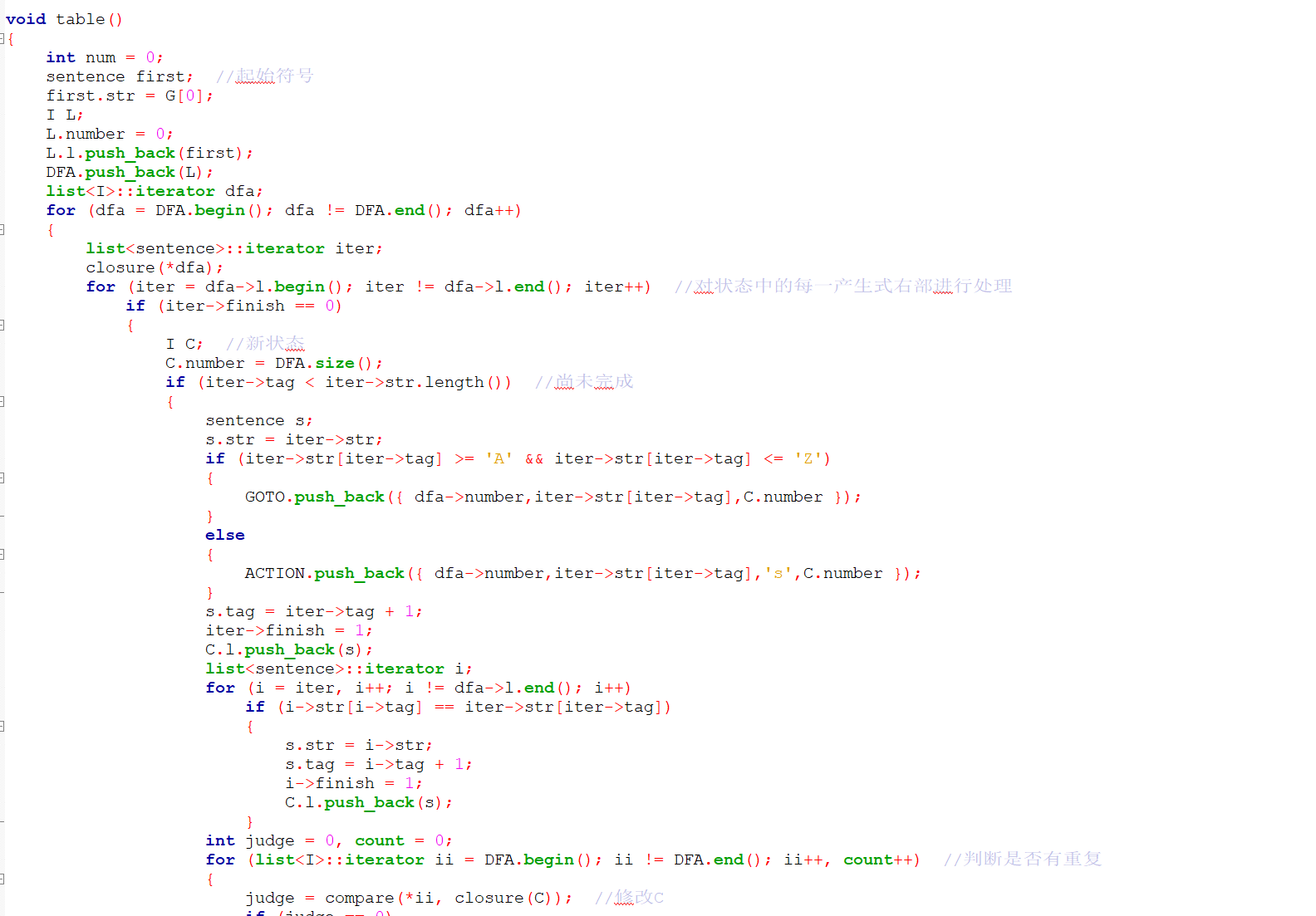


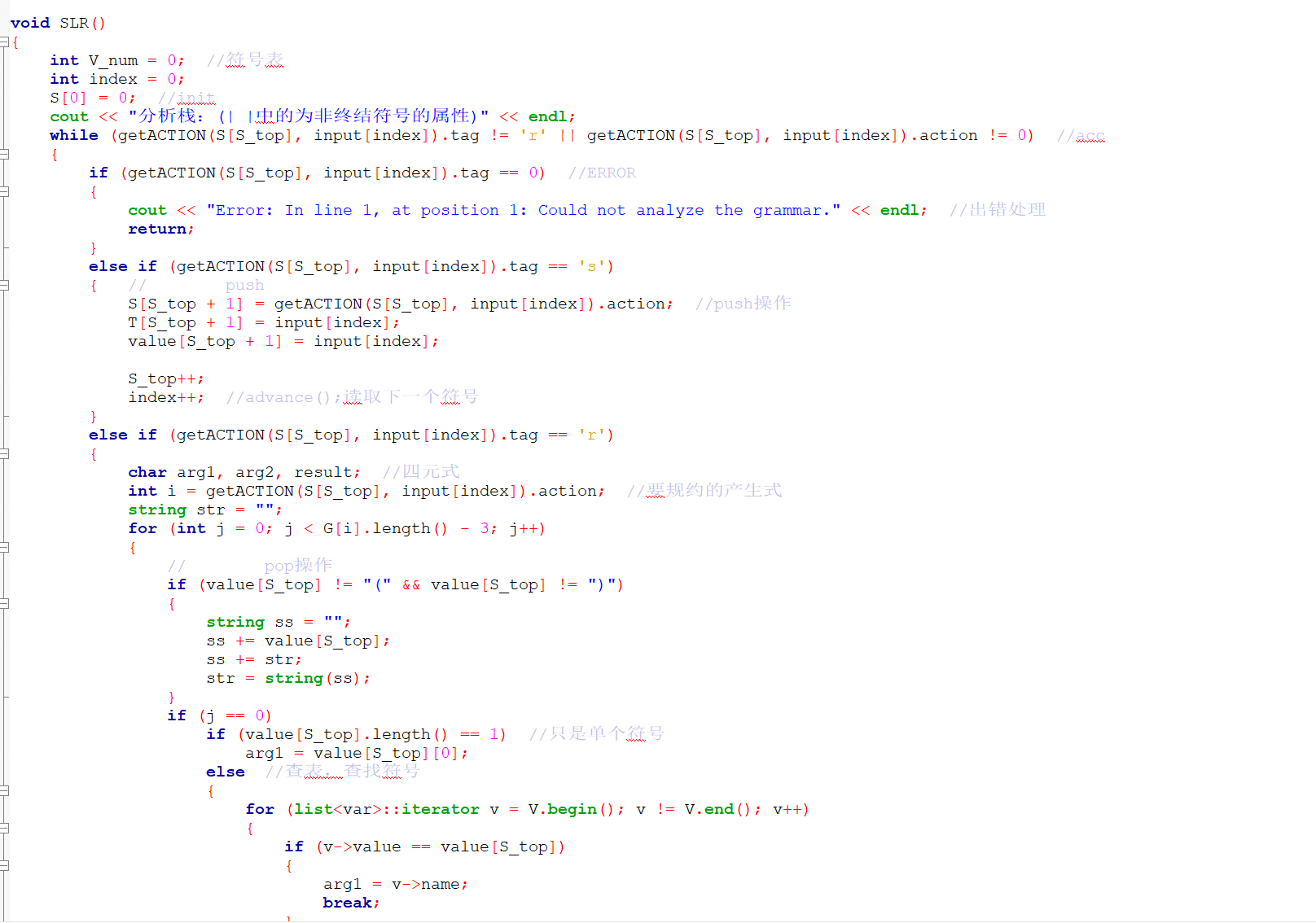
2、程序功能块

1、读取文件(获取输入的测试用例、得到表达式)

  
2.扫描文法(构建终结符号、非终结符号集、First/Follow集)

  
3.造表(构造项目集族、构造ACTION/GOTO、设置四元式)

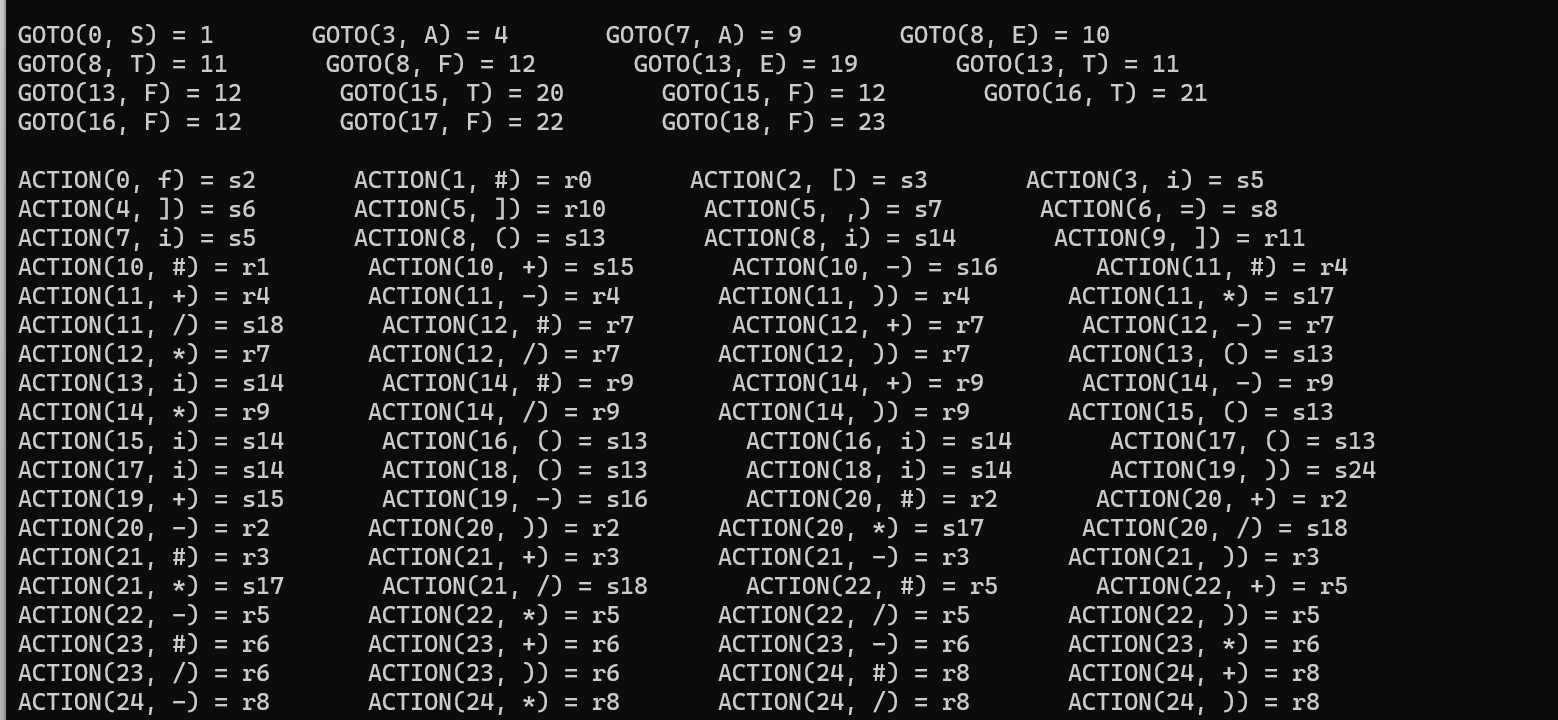
  
4.SLR分析(判断是否可以被分析程序接受、生成四元式)



1. **实验结果及测试**

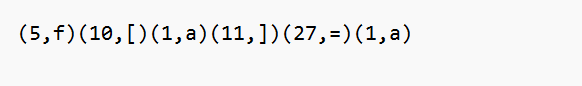
FIRST集、FOLLOW集和状态表：



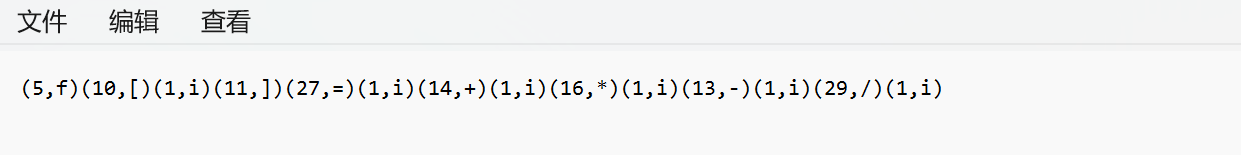
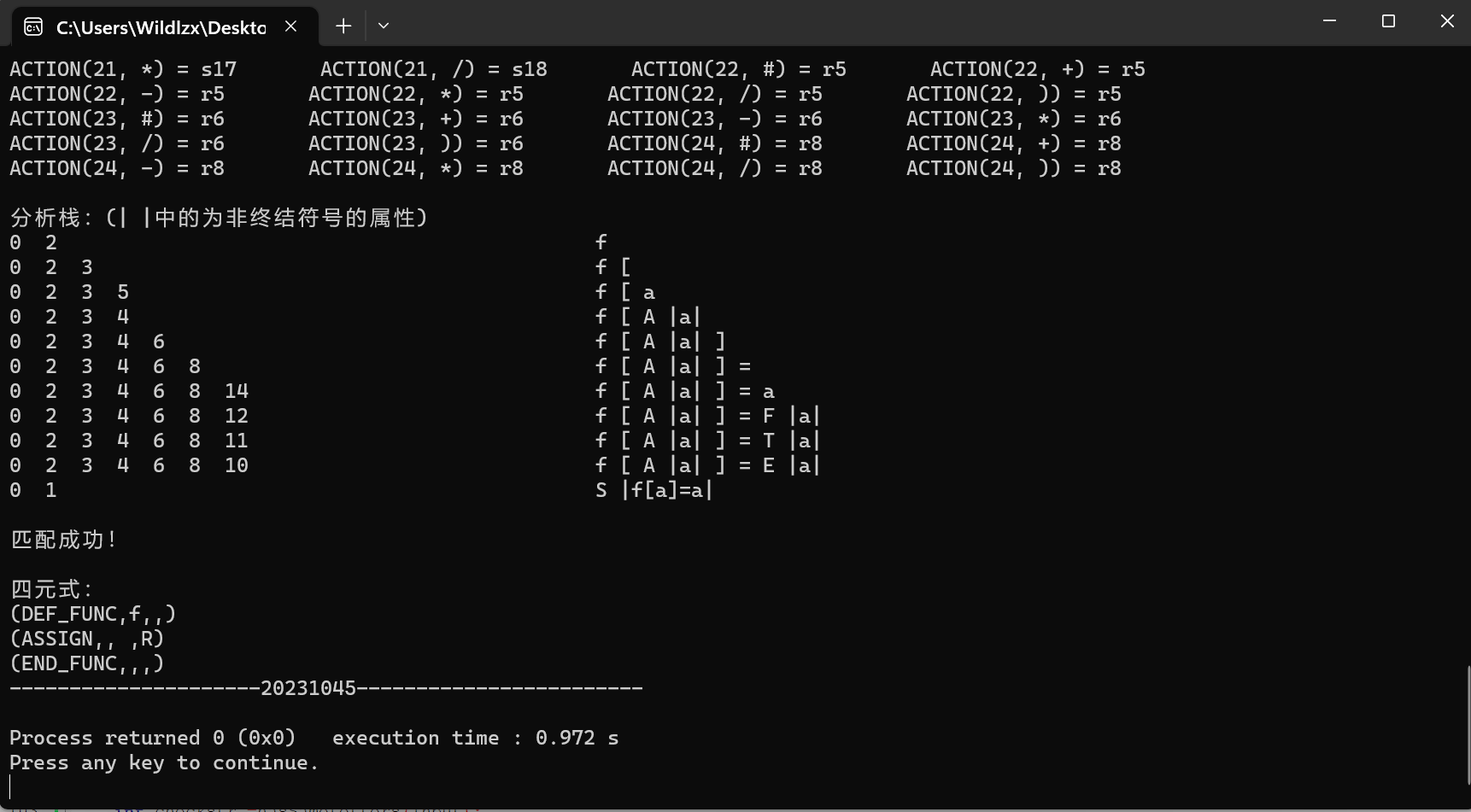
SLR(1)分析表：

正确测试样例

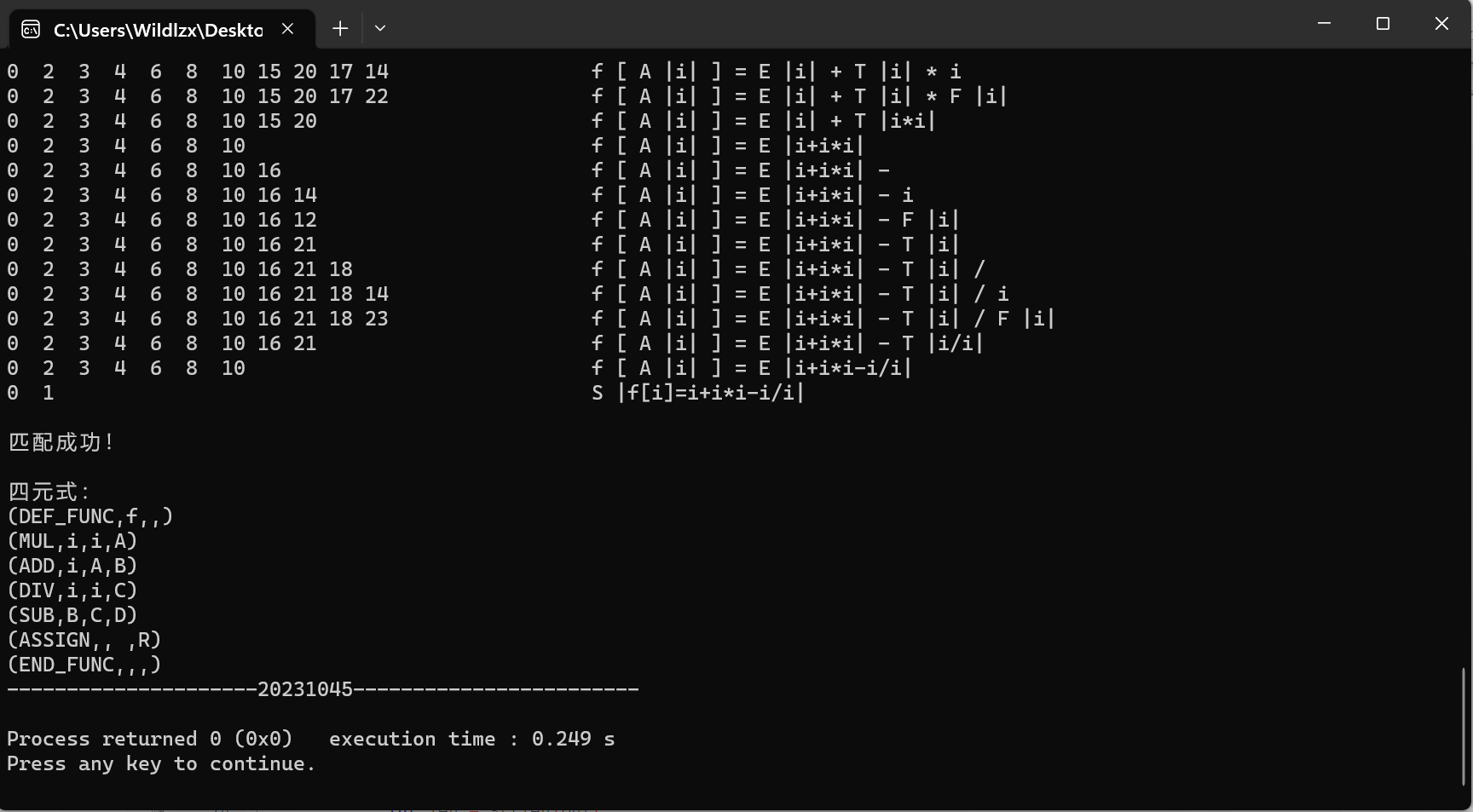
1、

输入：

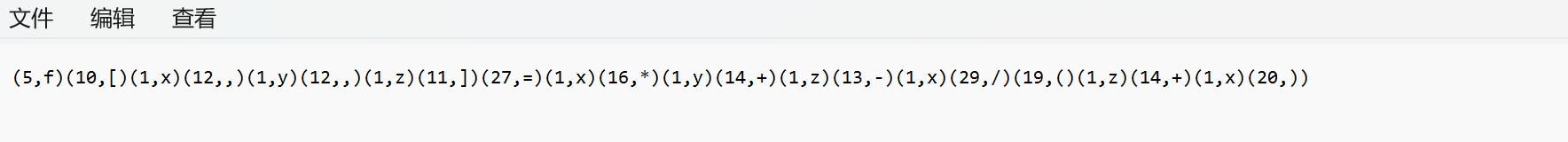
输出：

2、输入：

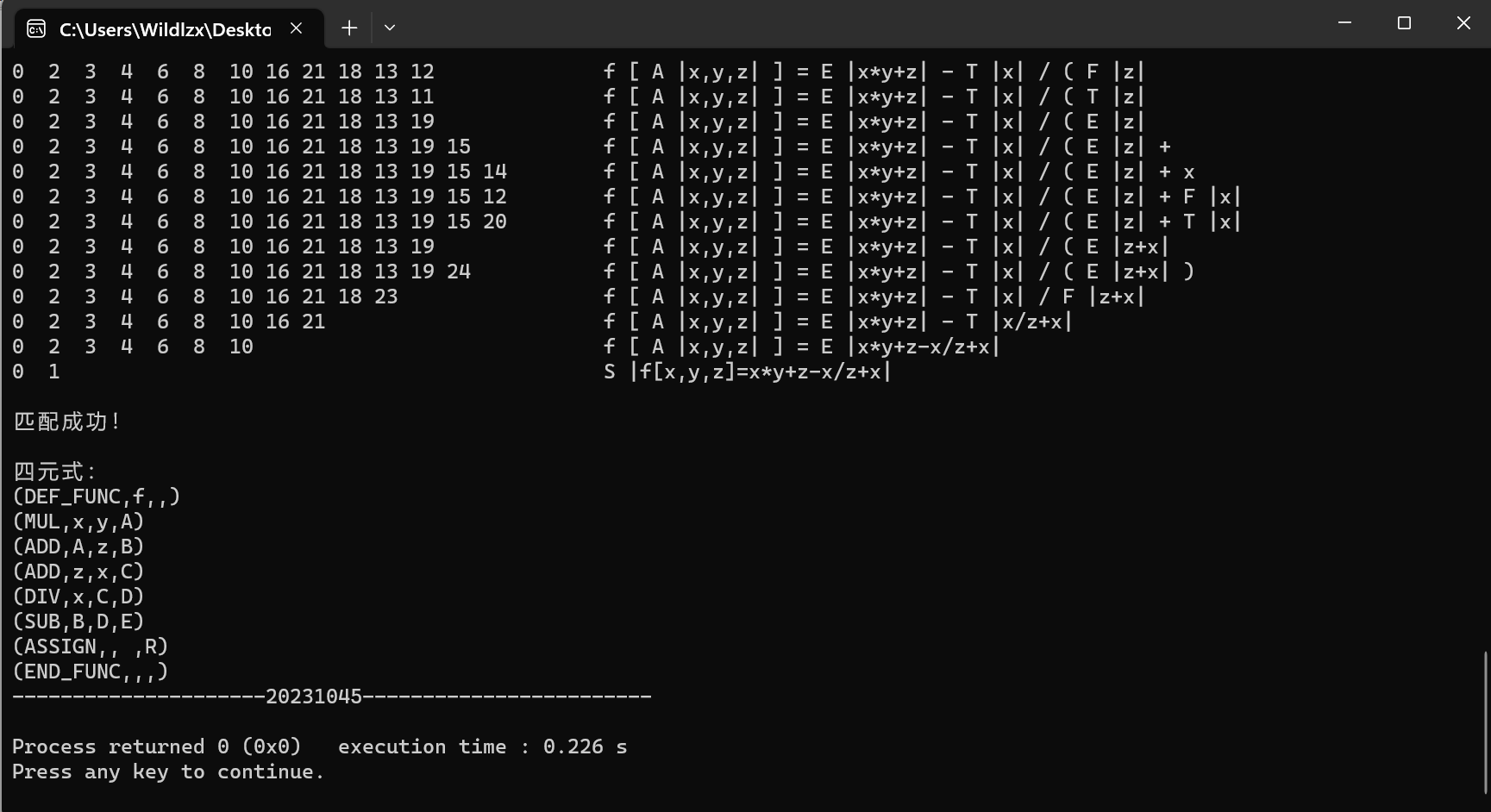
”

输出：

3输入：

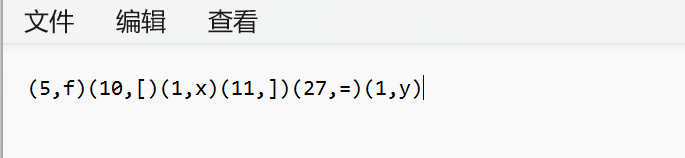


输出：



错误测试样例：

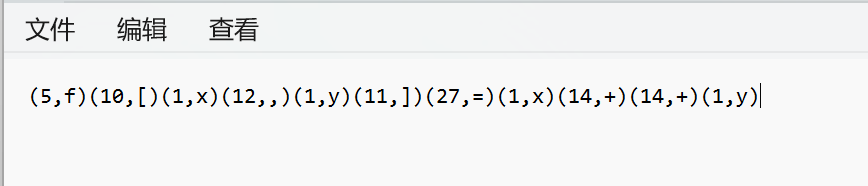
1. 输入：



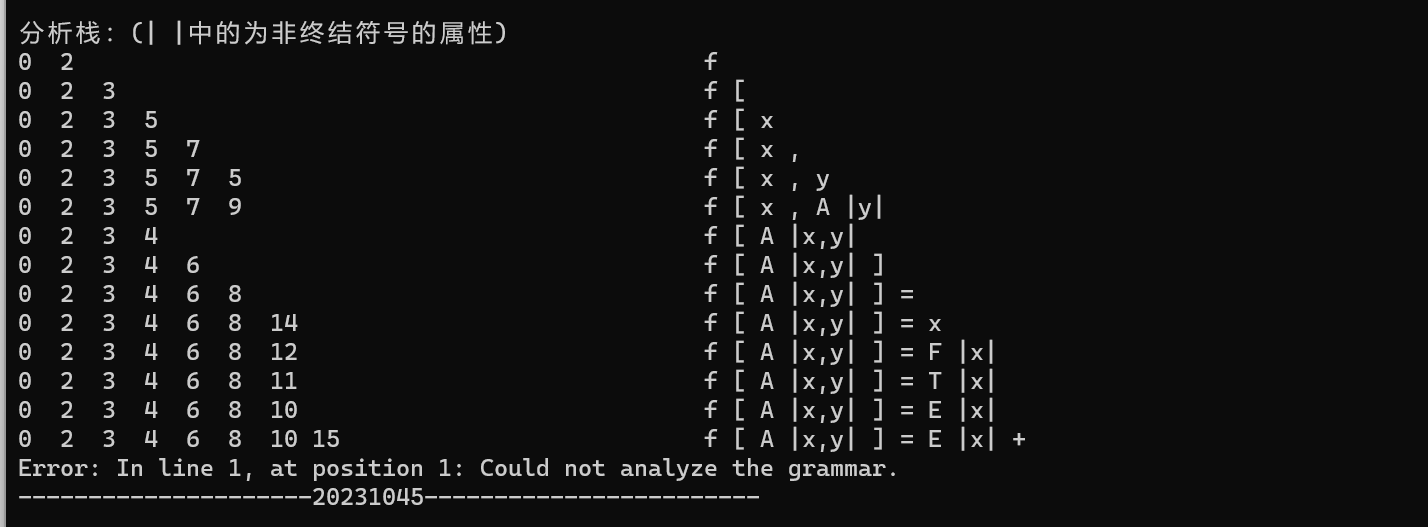
输出：



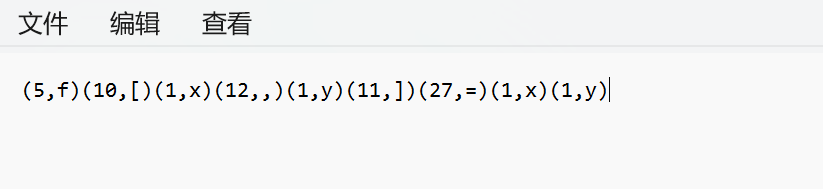
1. 输入：



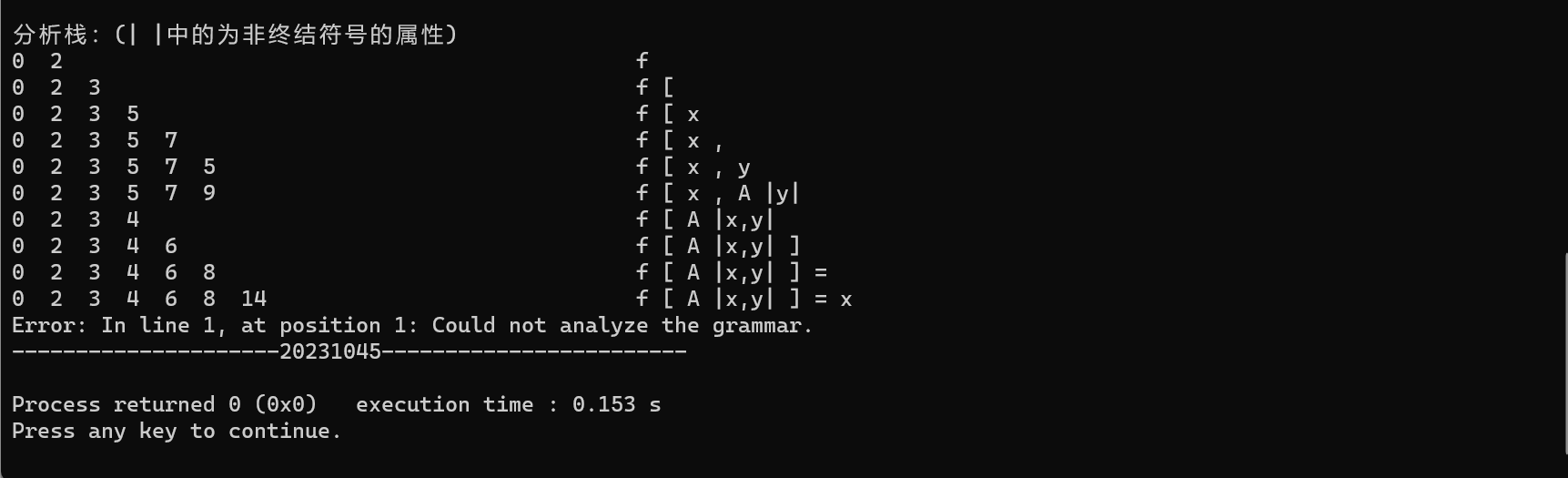
输出：



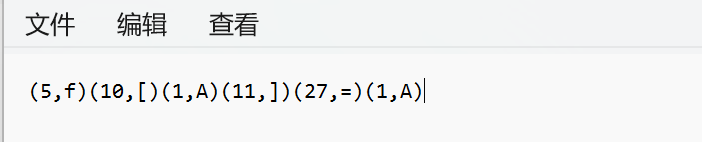
1. 输入：



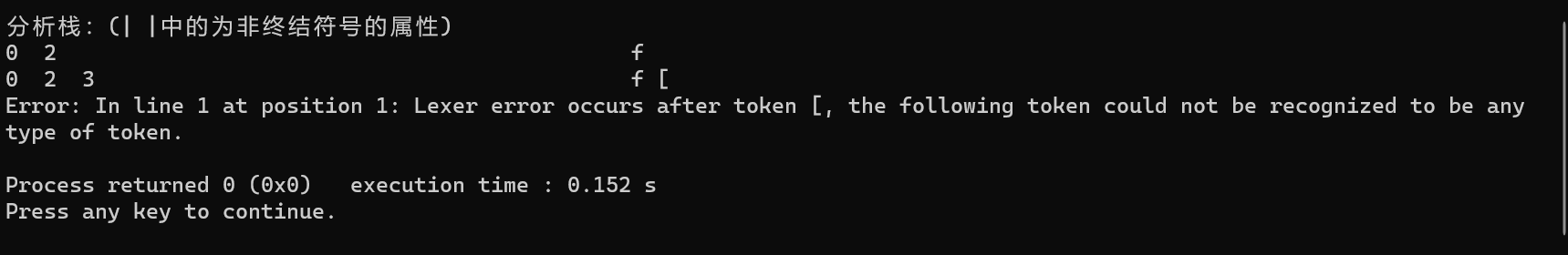
输出：



1. 输入：



输出：



1. **研究性教学实验感受**

刚开始写实验的时候有些无从下手，后面通过看书和查阅资料逐步了解需要完成任务和实现的方法，然后开始逐步实现应有的功能函数。通过这次实验让我对SLR(1)分析法的语法制导翻译及中间代码生成程序设计的原理的理解更深了一步，这也是做实验和学习这门课程的目的所在。一步步完善实验的过程也是在增强自己的知识。

**八、附件（源代码列表）**

#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include <string>

#include <list>

#include <stack>

#include <iostream>

#include<algorithm>

#include<fstream>

#include <bits/stdc++.h>

#include <unordered\_set>

using namespace std;

typedef struct s

{

int finish=0; //是否结束

int tag=3; //当前圆点的位置

string str;

}sentence;

typedef struct

{

int I; //当前状态

char Vt; //终结符号

char tag; //r||s

int action; //动作

}act;

typedef struct

{

int I; //当前状态

char Vn; //非终结符

int next\_I; //转移状态

}go;

typedef struct

{

int number; //编号

list<sentence> l;

}I; //状态

typedef struct

{

char op;

char arg1;

char arg2;

char result;

}siyuanshi;

typedef struct

{

char name;

string value; //place

}var; //变量

list<I> DFA; //状态集

list<char>\* First; //FIRST集

list<char>\* Follow; //FOLLOW集

list<act> ACTION; //ACTION[i][j]

list<go> GOTO; //GO[i][j]

list<char> Vt; //终结符号集

list<char> Vn; //非终结符号集

char input[100];//读取词法分析的输入

bool hasSameLetters(string str);

int readfile()

{

cout << "-------------------------------------------------------" << endl;

cout << "读取文件可知输入语句：" << endl;

FILE\* fp;

char buf[1000];

if ((fp = fopen("wrong4.txt", "r")) != NULL) {

while (fgets(buf, 1000, fp) != NULL)

{

int len = strlen(buf);

printf("%s \n", buf);

int flag = 0;

for (int i = 0; i < len; i++) {

if (buf[i] == '(' && flag == 0) {

flag = 1;//识别符号

for (int j = i + 1; j < len; j++) {

if (buf[j] == ',') {

input[strlen(input)] = buf[j + 1];

i = j + 1;

break;

}

}

}

else if (flag == 1 && buf[i] == ')') {

flag = 0;

}

}

}

}

input[strlen(input)] = '#';

fclose(fp);

cout << endl;

cout << endl << "输入串为：" << input << endl << endl;

int checkstr =hasSameLetters(input);

if (checkstr) {

// cout << "Equal sides contain same letters" << endl;

return 3;

} else {

//cout << "Equal sides do not contain same letters" << endl;

return 2;

}

//return 0;

}

int S[100]; //分析栈

int S\_top = 0; //分析栈栈顶

char T[100]; //符号栈 栈顶与分析栈总相同

string value[100]; //存储符号的属性

list<siyuanshi> L\_sys; //四元式

list<int> sys; //产生四元式的状态集

list<var> V; //变量表

int result\_count = 0;

//string G[11] = { "S->A", "A->V=E","E->E+T","E->E-T","E->T","T->T\*F","T->T/F","T->F","F->(E)","F->i","V->i" };

string G[12] = { "V->S","S->f[A]=E","E->E+T","E->E-T","E->T","T->T\*F","T->T/F","T->F","F->(E)","F->i","A->i","A->i,A"};

act getACTION(int i, char b)

{

char c = b;

if (b >= 'a' && b <= 'z'&&b!='f')

b = 'i';

if(b=='f')

b='f';

if(b >= 'A' && b <= 'Z'){

cout<<"Error: In line 1 at position 1: Lexer error occurs after token [, the following token could not be recognized to be any type of token."<<endl;

exit(0);}

list<act>::iterator it;

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

{

if (it->I == i && it->Vt == b)

{

return \*it;

}

}

return { 0,0,0,0 }; //ERROR

}

int getGOTO(int i, char b)

{

list<go>::iterator it;

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

{

if (it->I == i && it->Vn == b)

return it->next\_I;

}

return -1; //ERROR

}

char newTemp()

{

char c = 'A';

result\_count++;

return c + result\_count - 1;

}

void SLR()

{

int V\_num = 0; //符号表

int index = 0;

S[0] = 0; //init

cout << "分析栈：(| |中的为非终结符号的属性)" << endl;

while (getACTION(S[S\_top], input[index]).tag != 'r' || getACTION(S[S\_top], input[index]).action != 0) //acc

{

if (getACTION(S[S\_top], input[index]).tag == 0) //ERROR

{

cout << "Error: In line 1, at position 1: Could not analyze the grammar." << endl; //出错处理

return;

}

else if (getACTION(S[S\_top], input[index]).tag == 's')

{ // push

S[S\_top + 1] = getACTION(S[S\_top], input[index]).action; //push操作

T[S\_top + 1] = input[index];

value[S\_top + 1] = input[index];

S\_top++;

index++; //advance();读取下一个符号

}

else if (getACTION(S[S\_top], input[index]).tag == 'r')

{

char arg1, arg2, result; //四元式

int i = getACTION(S[S\_top], input[index]).action; //要规约的产生式

string str = "";

for (int j = 0; j < G[i].length() - 3; j++)

{

// pop操作

if (value[S\_top] != "(" && value[S\_top] != ")")

{

string ss = "";

ss += value[S\_top];

ss += str;

str = string(ss);

}

if (j == 0)

if (value[S\_top].length() == 1) //只是单个符号

arg1 = value[S\_top][0];

else //查表，查找符号

{

for (list<var>::iterator v = V.begin(); v != V.end(); v++)

{

if (v->value == value[S\_top])

{

arg1 = v->name;

break;

}

}

}

if (j == 2)

if (value[S\_top].length() == 1)

arg2 = value[S\_top][0];

else

for (list<var>::iterator v = V.begin(); v != V.end(); v++)

{

if (v->value == value[S\_top])

{

arg2 = v->name;

break;

}

}

S[S\_top] = -1;

T[S\_top] = 0;

value[S\_top] = "";

S\_top--;

}

for (list<int>::iterator ii = sys.begin(); ii != sys.end(); ii++) //生成四元式

if (i == \*ii)

{

result = newTemp();

V.push\_back({ result,str });

/\*if(G[i][4]=='+')

L\_sys.push\_back({ 'ADD',arg1,arg2,result });

if(G[i][4]=='-')

L\_sys.push\_back({ 'SUB',arg1,arg2,result });

if(G[i][4]=='\*')

L\_sys.push\_back({ 'MUL',arg1,arg2,result });

if(G[i][4]=='/')

L\_sys.push\_back({ 'DIV',arg1,arg2,result });

if(G[i][4]=='=')

L\_sys.push\_back({ 'ASSIGN',arg1,arg2,result });\*/

L\_sys.push\_back({ G[i][4],arg1,arg2,result });

break;

}

//push

S[S\_top + 1] = getGOTO(S[S\_top], G[i][0]);

T[S\_top + 1] = G[i][0];

value[S\_top + 1] = str; //属性传递

S\_top++;

}

for (int tt = 0; tt <= 15; tt++)

{

if (tt <= S\_top)

if (S[tt] >= 10)

cout << S[tt] << " ";

else

cout << S[tt] << " ";

else

cout << " ";

}

for (int tt = 0; tt <= S\_top; tt++)

{

if (tt <= S\_top)

{

cout << T[tt] << " ";

if (isupper(T[tt]))

cout << "|" << value[tt] << "| ";

}

else

cout << " ";

}

cout << endl;

}

cout << endl << "匹配成功！" << endl;

cout << endl << "四元式：" << endl;

cout << "(DEF\_FUNC,f,,)" << endl;

for (list<siyuanshi>::iterator s = L\_sys.begin(); s != L\_sys.end(); s++)

{

if(s->op=='+')

cout << '(' << "ADD" << "," << s->arg2 << "," << s->arg1 << ',' << s->result << ')' << endl;

if(s->op=='-')

cout << '(' << "SUB" << "," << s->arg2 << "," << s->arg1 << ',' << s->result << ')' << endl;

if(s->op=='\*')

cout << '(' << "MUL" << "," << s->arg2 << "," << s->arg1 << ',' << s->result << ')' << endl;

if(s->op=='/')

cout << '(' << "DIV" << "," << s->arg2<< "," << s->arg1 << ',' << s->result << ')' << endl;

//if(s == L\_sys.end())

//cout << '(' << "ASSIGN" << "," << s->result << "," << " " << ',' << "R" << ')' << endl;

//cout << '(' << s->op << "," << s->arg1 << "," << s->arg2 << ',' << s->result << ')' << endl;

}

list<siyuanshi>::iterator s = L\_sys.end();

cout << '(' << "ASSIGN" << "," << s->result << "," << " " << ',' << "R" << ')' << endl;

cout << "(END\_FUNC,,,)" << endl;

}

void print(sentence s)

{

int r;

cout << " ";

for (r = 0; r < s.str.length(); r++)

if (r == s.tag)

cout << "·" << s.str[r];

else

cout << s.str[r];

if (r == s.tag)

cout << "·";

cout << endl;

}

void find(char ch, int tag\_Vn)

{

for (int qq = 0; qq < 12; qq++)

{

if (ch == G[qq][0])

{

if (G[qq][3] >= 'A' && G[qq][3] <= 'Z')

if (G[qq][3] == ch) //避免死循环重复查找

;

else

find(G[qq][3], tag\_Vn);

else

{

int tag = 0;

for (list<char>::iterator i = First[tag\_Vn].begin(); i != First[tag\_Vn].end(); i++)

{

if (\*i == G[qq][3])

{

tag = 1;

break;

}

}

if (tag == 0)

First[tag\_Vn].push\_back(G[qq][3]);

}

}

}

}

int findVn(char c)

{

int tag = -1;

for (list<char>::iterator i = Vn.begin(); i != Vn.end(); i++)

{

tag++;

if (c == \*i)

return tag;

}

return -1;

}

void scan() //构造符号集

{

for (int i = 0; i < 12; i++)//文法长度

{

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

{

if (j == 1 || j == 2)

continue;

if (isupper(G[i][j]))

{

if (find(Vn.begin(), Vn.end(), G[i][j]) == Vn.end())

Vn.push\_back(G[i][j]);

}

else

{

if (find(Vt.begin(), Vt.end(), G[i][j]) == Vt.end())

Vt.push\_back(G[i][j]);

}

}

}

Vt.push\_back('#');

First = new list<char>[Vn.size()];

int tag\_Vn = -1;

for (list<char>::iterator ch = Vn.begin(); ch != Vn.end(); ch++)

{

tag\_Vn++;

for (int qq = 0; qq < 12; qq++)

{

if (\*ch == G[qq][0])

{

if (G[qq][3] >= 'A' && G[qq][3] <= 'Z')

{

find(G[qq][3], tag\_Vn);

}

else

{

First[tag\_Vn].push\_back(G[qq][3]);

}

}

}

}

Follow = new list<char>[Vn.size()];

Follow[0].push\_back('#');

int follow\_sum = 0; //follow集数量总和

int previous\_sum = -1; //之前的总和

while (follow\_sum != previous\_sum)

{

for (int i = 0; i < 12; i++)

{

for (int j = 3; j < G[i].length() - 1; j++)

{

if (isupper(G[i][j]))

if (!isupper(G[i][j + 1])) //是终结符号

{

int tag = 0;

for (list<char>::iterator ii = Follow[findVn(G[i][j])].begin(); ii != Follow[findVn(G[i][j])].end(); ii++)

{

if (\*ii == G[i][j + 1])

{

tag = 1;

break;

}

}

if (tag == 0)

Follow[findVn(G[i][j])].push\_back(G[i][j + 1]);

}

else //是非终结符号

{

for (list<char>::iterator ii = Follow[findVn(G[i][j + 1])].begin(); ii != Follow[findVn(G[i][j + 1])].end(); ii++)

{

if (find(Follow[findVn(G[i][j + 1])].begin(), Follow[findVn(G[i][j + 1])].end(), \*ii) == Follow[findVn(G[i][j + 1])].end()) //不重复

Follow[findVn(G[i][j])].push\_back(\*ii);

}

}

}

if (isupper(G[i][G[i].length() - 1]))

for (list<char>::iterator ii = Follow[findVn(G[i][0])].begin(); ii != Follow[findVn(G[i][0])].end(); ii++)

{

if (find(Follow[findVn(G[i][G[i].length() - 1])].begin(), Follow[findVn(G[i][G[i].length() - 1])].end(), \*ii) == Follow[findVn(G[i][G[i].length() - 1])].end()) //不重复

Follow[findVn(G[i][G[i].length() - 1])].push\_back(\*ii);

}

}

previous\_sum = follow\_sum;

follow\_sum = 0;

for (int ii = 0; ii < Vn.size(); ii++)

{

follow\_sum += Follow[ii].size();

}

}

list<char>::iterator iter;

cout << "终结符号集：";

for (iter = Vt.begin(); iter != Vt.end(); iter++)

{

cout << \*iter;

}

cout << endl;

cout << "非终结符号集：";

for (iter = Vn.begin(); iter != Vn.end(); iter++)

{

cout << \*iter;

}

cout << endl;

int pp;

cout << "FIRST" << endl;

for (pp = 0, iter = Vn.begin(); pp < Vn.size(); pp++, iter++)

{

cout << \*iter << " ";

for (list<char>::iterator ch = First[pp].begin(); ch != First[pp].end(); ch++)

{

cout << \*ch << " ";

}

cout << endl;

}

cout << "FOLLOW" << endl;

for (pp = 0, iter = Vn.begin(); pp < Vn.size(); pp++, iter++)

{

cout << \*iter << " ";

for (list<char>::iterator ch = Follow[pp].begin(); ch != Follow[pp].end(); ch++)

{

cout << \*ch << " ";

}

cout << endl;

}

}

I closure(I& dfa)

{

list<sentence>::iterator iter;

for (iter = dfa.l.begin(); iter != dfa.l.end(); iter++)

{

if ((iter->tag <= iter->str.length()))

if ((iter->str[iter->tag] >= 'A' && iter->str[iter->tag] <= 'Z'))

{

for (int tt = 0; tt < 12; tt++) //查找对应的产生式

{

if (iter->str[iter->tag] == G[tt][0])

{

int q = 0;

list<sentence>::iterator it;

sentence x;

x.str = G[tt];

for (it = dfa.l.begin(); it != dfa.l.end(); it++)

{

if ((it->str == x.str) && (it->tag == x.tag))

{

q = 1;

break;

}

}

if (q == 0)

{

dfa.l.push\_back(x);

}

}

}

}

}

return dfa;

}

int compare(I a, I b) //是否相等

{

if (a.l.size() != b.l.size())

return 1;

else

{

for (list<sentence>::iterator iii = a.l.begin(), jjj = b.l.begin(); iii != a.l.end() || jjj != b.l.end(); iii++, jjj++)

if (iii->str != jjj->str || iii->tag != jjj->tag) //判断是否相等

{

return 1;

}

}

return 0;

}

void table()

{

int num = 0;

sentence first; //起始符号

first.str = G[0];

I L;

L.number = 0;

L.l.push\_back(first);

DFA.push\_back(L);

list<I>::iterator dfa;

for (dfa = DFA.begin(); dfa != DFA.end(); dfa++)

{

list<sentence>::iterator iter;

closure(\*dfa);

for (iter = dfa->l.begin(); iter != dfa->l.end(); iter++) //对状态中的每一产生式右部进行处理

if (iter->finish == 0)

{

I C; //新状态

C.number = DFA.size();

if (iter->tag < iter->str.length()) //尚未完成

{

sentence s;

s.str = iter->str;

if (iter->str[iter->tag] >= 'A' && iter->str[iter->tag] <= 'Z')

{

GOTO.push\_back({ dfa->number,iter->str[iter->tag],C.number });

}

else

{

ACTION.push\_back({ dfa->number,iter->str[iter->tag],'s',C.number });

}

s.tag = iter->tag + 1;

iter->finish = 1;

C.l.push\_back(s);

list<sentence>::iterator i;

for (i = iter, i++; i != dfa->l.end(); i++)

if (i->str[i->tag] == iter->str[iter->tag])

{

s.str = i->str;

s.tag = i->tag + 1;

i->finish = 1;

C.l.push\_back(s);

}

int judge = 0, count = 0;

for (list<I>::iterator ii = DFA.begin(); ii != DFA.end(); ii++, count++) //判断是否有重复

{

judge = compare(\*ii, closure(C)); //修改C

if (judge == 0)

break;

}

if (judge == 0)

{

if (iter->str[iter->tag] >= 'A' && iter->str[iter->tag] <= 'Z')

{

GOTO.pop\_back();

GOTO.push\_back({ dfa->number,iter->str[iter->tag], count });

}

else

{

ACTION.pop\_back();

ACTION.push\_back({ dfa->number,iter->str[iter->tag],'s',count });

}

}

else

DFA.push\_back(C);

}

else //已经完成

{

int cc = 0, tt;

for (tt = 0; tt < 12; tt++)

{

if (iter->str == G[tt])

break;

}

for (list<char>::iterator c = Vn.begin(); c != Vn.end(); c++, cc++)

if (\*c == iter->str[0])

break;

for (list<char>::iterator c = Follow[cc].begin(); c != Follow[cc].end(); c++)

{

ACTION.push\_back({ dfa->number,\*c,'r',tt }); //对应的第j条产生式

for (int ss = 3; ss < G[tt].length(); ss++) //查找四元式对应的产生式

if (G[tt][ss] == '+' || G[tt][ss] == '-' || G[tt][ss] == '\*' || G[tt][ss] == '/')

{

sys.push\_back(tt);

}

}

}

}

}

num = 0;

for (dfa = DFA.begin(); dfa != DFA.end(); dfa++)

{

cout << "状态" << num++ << "：" << endl;

for (list<sentence>::iterator iii = dfa->l.begin(); iii != dfa->l.end(); iii++)

{

cout << " ";

print(\*iii);

}

}

cout << endl;

int kkk = 0;

for (list<go>::iterator g = GOTO.begin(); g != GOTO.end(); g++)

{

cout << "GOTO(" << g->I << ", " << g->Vn << ") = " << g->next\_I<<" ";

kkk++;

if (kkk % 4 == 0) {

cout << endl;

}

}

cout << endl << endl;

kkk = 0;

for (list<act>::iterator a = ACTION.begin(); a != ACTION.end(); a++)

{

cout << "ACTION(" << a->I << ", " << a->Vt << ") = " << a->tag << a->action << " ";

kkk++;

if (kkk % 4 == 0) {

cout << endl;

}

}

cout << endl;

}

int main()

{

//readfile();

int p;

p=readfile();

scan();

table();

if(p==2){

cout << "Error: Used undeclared variable y" <<endl;

return 0;}

SLR();

cout << "---------------------20231045------------------------" << endl;

return 0;

}

bool hasSameLetters(string str) {

unordered\_set<char> left, right;

int equals\_index = str.find("=");

for (int i = 0; i < equals\_index; i++) {

left.insert(str[i]);

}

for (int i = equals\_index + 1; i < str.length(); i++) {

right.insert(str[i]);

}

for (auto letter : left) {

if (right.count(letter) > 0) {

return true;

}

}

return false;

}