**编译原理实验报告**

实验名称：\_\_\_\_\_\_\_\_语法制导翻译\_\_\_\_\_\_\_\_

指导教师：\_\_\_\_\_\_\_\_\_王莹洁\_\_\_\_\_\_\_\_\_\_\_\_

专业班级：\_\_\_\_\_\_\_\_计198-2\_\_\_\_\_\_\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_\_\_\_\_\_刘鑫\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

学号：\_\_\_\_\_\_\_\_\_201958508239\_\_\_\_\_\_\_\_\_\_\_

电子邮件\_\_\_\_1579664489@qq.com\_\_\_\_\_\_\_\_\_

实验地点：\_\_\_\_科技馆5405\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

实验成绩：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

日期： 2021年 12月 27 日

**一、实验题目**

在对简化的算术表达式进行语法分析的同时生成四元式。

**二、实验内容**

程序输入/输出示例（以下仅供参考）：  
对下列文法，用SLR（1）分析法对任意输入的符号串进行分析，并输出对于的四元式：

(0)S'→E

(1)E→E1+T

(2)E→T

(3)T→T\*F

(4)T→F

(5)F→(E)

(6)F→i

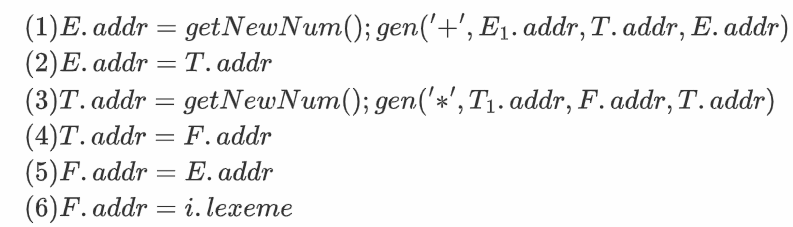
  输出的格式如下：  
  (1) SLR(1)分析程序并输出四元式，编制人：姓名，学号，班级  
  (2)输入一个以#结束的符号串(包括+—\*/（）i#)：在此位置输入符号串  
  (3)输出过程如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 步骤 | 状态栈 | 符号栈 | 剩余输入串 | ACTION | GOTO | 四元式 |
| 1 | 0 | # | i+i\*i# | R5 |  |  |

**三、实验步骤**

（包括基本设计思路、流程框图、算法设计、函数相关说明、输入与输出以及程序运行结果）

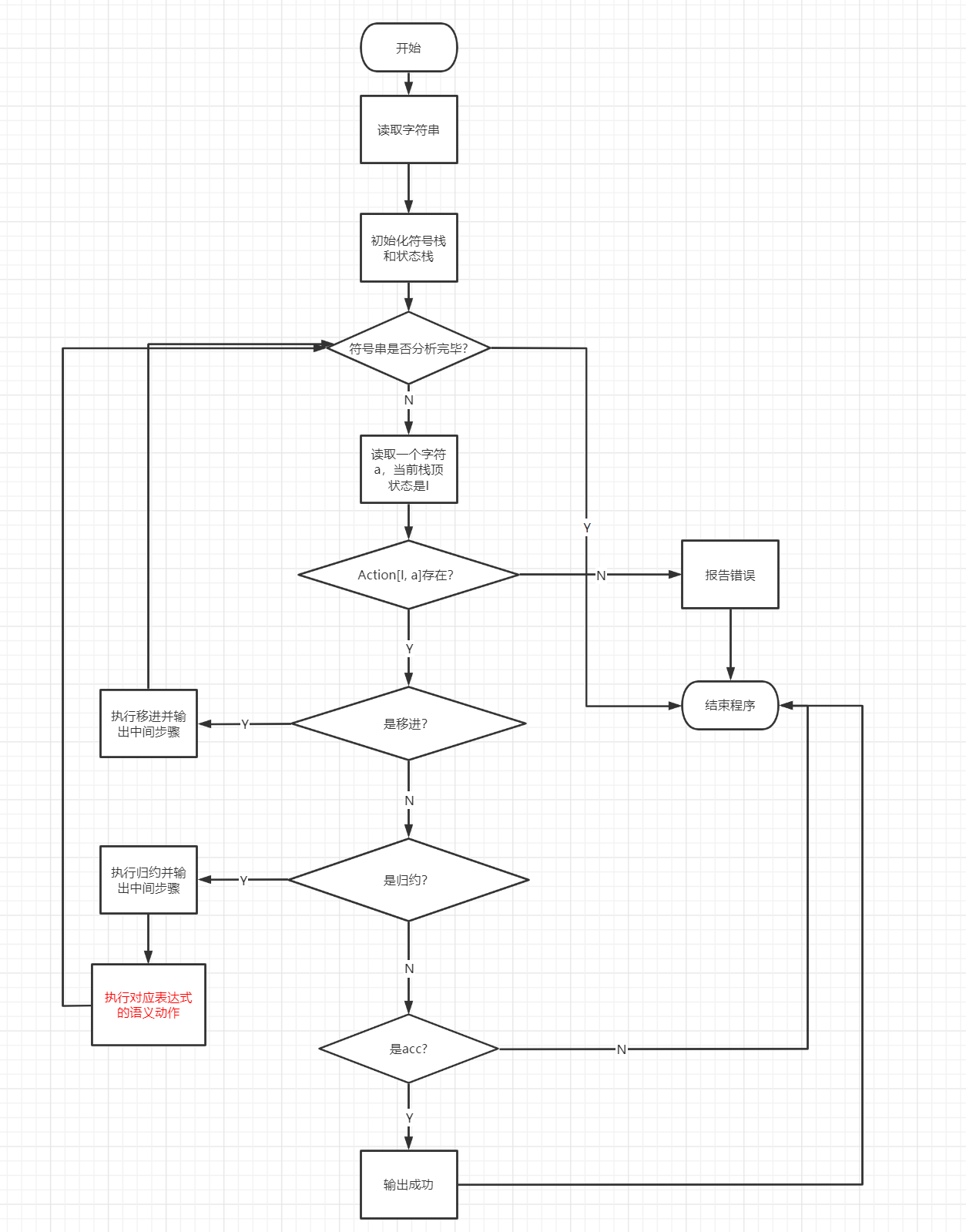
基本设计思路：为原始文法添加如下语义规则



其中getNewNum()表示生成一个新的临时变量符号,gen表示生成四元式,addr属性是一个综合属性，存放对于非终结符的临时变量,lexeme表示终结符的词素。

在自底向上的语法分析过程中，只需要在执行归约过程中，运行表达式相应的语义规则即可完成对四元式的翻译。

流程框图(与实验三SLR1语法分析相同，不同之处在于归约中会执行语义动作)：



函数相关说明：

init:初始化状态站和符号栈，初始化当前输入字符指针和步骤数记录

error:输出错误

printStep:输出中间步骤

getPos:获取终结符或非终结符对应的索引

shift:进行移进操作

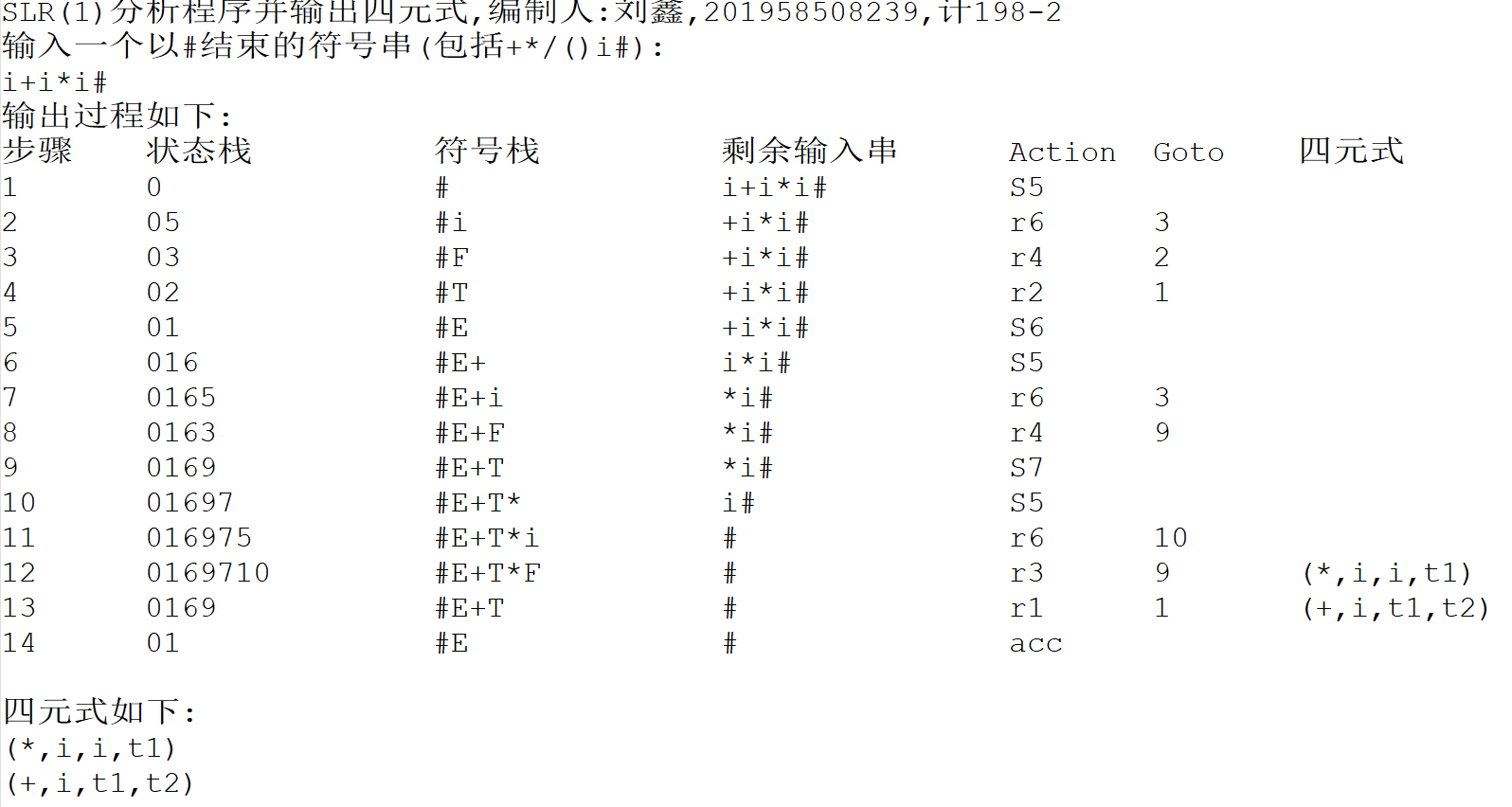
reduction:执行归约

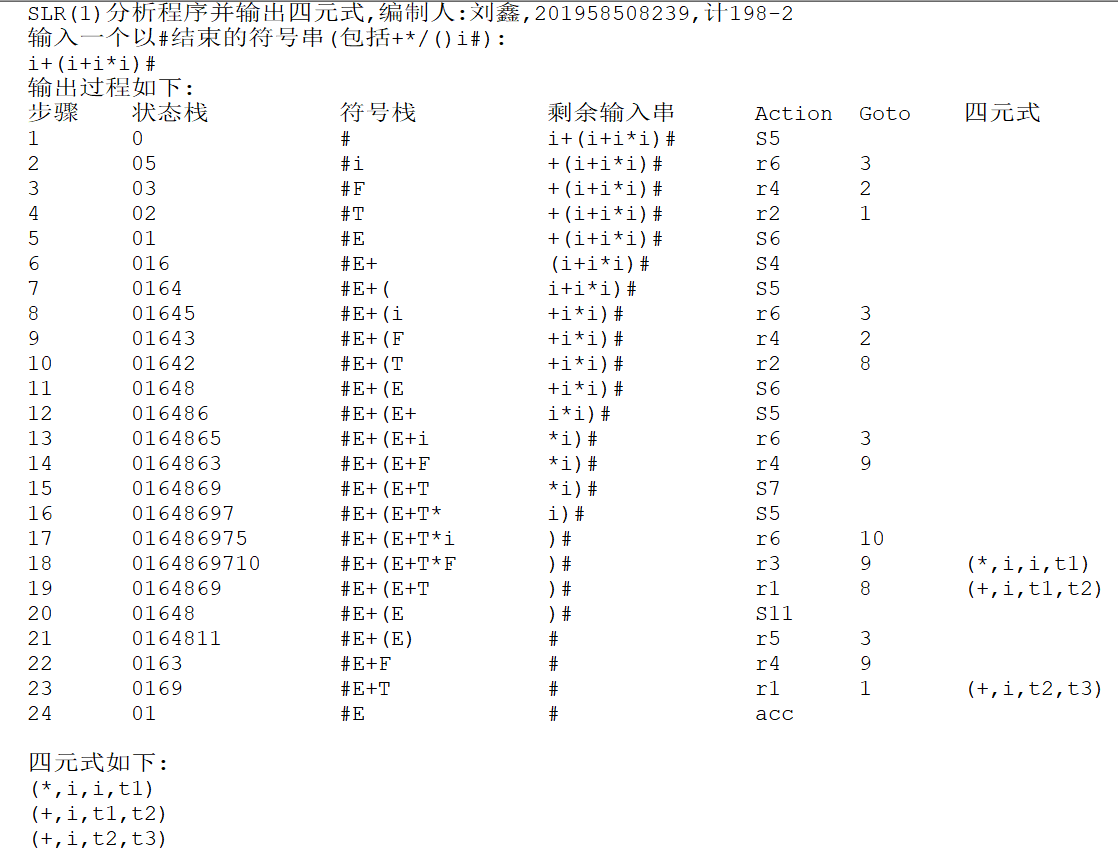
getNewNum:获取新的临时变量符号

makeTAC:执行表达式对应的语义动作生成四元式

analyse:入口函数，根据输入的字符串进行SLR(1)分析

程序运行结果：





1. **实验心得体会**

在本次实验中，通过在实验三原有的文法基础上，通过添加语义规则即可生成对应的四元式，过程相对简单，只要对自底向上的语法分析过程掌握熟练，不难看出在应该在归约的过程执行对应的语义规则，因为归约的过程要用表达式的左部去替换表达式的右部，这可以理解为要进行一次运算，因此要对应与四元式的生成，但是并不是所有的表达式都是运算，因此有一部分表达式只有计算综合属性的语义规则代码；只有那些真正代表右部含运算符的表达式才要生成四元式，生成前需要为符号创建一个新的临时变量。只要对上述生成规则稍加改造即可生成三元式。

在具体编写中，一直出现无法输出无法对齐现象，经过反复查找，发现原来\t并不一定代表固定的空格数量，它与前面的字符个数有关，因此\t并不适合做对齐操作，这也是以前编程中没有注意到的一个小问题。

**五、源程序清单（代码）**

#include <iostream>

#include <cstdio>

#include <cstring>

#include <string>

#include <vector>

#include <stack>

#include <sstream>

#include <cstdlib>

using namespace std;

class SLR1

{

private:

struct Node

{

char symbol;

string addr;

explicit Node(char sy, string addr = "")

{

this->symbol = sy;

this->addr = addr;

}

};

string m\_actionTable[12][6] = {"S5", "0", "0", "S4", "0", "0",

"0", "S6", "0", "0", "0", "acc",

"0", "r2", "S7", "0", "r2", "r2",

"0", "r4", "r4", "0", "r4", "r4",

"S5", "0", "0", "S4", "0", "0",

"0", "r6", "r6", "0", "r6", "r6",

"S5", "0", "0", "S4", "0", "0",

"S5", "0", "0", "S4", "0", "0",

"0", "S6", "0", "0", "S11", "0",

"0", "r1", "S7", "0", "r1", "r1",

"0", "r3", "r3", "0", "r3", "r3",

"0", "r5", "r5", "0", "r5", "r5"};

int m\_gotoTable[12][3] = {1, 2, 3,

0, 0, 0,

0, 0, 0,

0, 0, 0,

8, 2, 3,

0, 0, 0,

0, 9, 3,

0, 0, 10,

0, 0, 0,

0, 0, 0,

0, 0, 0,

0, 0, 0};

string m\_VT = "i+\*()#";

string m\_VN = "ETF";

vector<string> m\_production = {"E->E+T", "E->T", "T->T\*F", "T->F", "F->(E)", "F->i"};

vector<int> m\_stateStack;

vector<Node> m\_symbolStack;

vector<string> m\_tac;

int m\_curPos;

int m\_step;

string m\_target;

char m\_errorMsg[100];

void init(string &s)

{

m\_target = s;

m\_stateStack.push\_back(0);

m\_symbolStack.push\_back(Node('#'));

m\_curPos = 0;

m\_step = 1;

}

void error()

{

cout << "Error\t" << m\_errorMsg << endl;

exit(0);

}

void printStep(int stateNum, int flag = 0)

{

cout << m\_step++ << "\t";

for (int i = 0; i < m\_stateStack.size(); i++)

cout << m\_stateStack[i];

cout << "\t\t";

for (int i = 0; i < m\_symbolStack.size(); i++)

cout << m\_symbolStack[i].symbol;

cout << "\t\t";

cout << m\_target.substr(m\_curPos) << "\t";

if (flag == 0)

cout << "\tS" << stateNum << endl;

else if (flag == 1)

{

// 执行归约

reduction(stateNum);

cout << "\tr" << stateNum << "\t";

if (m\_stateStack.back() == 0)

error();

else

cout << m\_stateStack.back() << "\t";

if (stateNum == 1 || stateNum == 3)

cout << m\_tac.back();

cout << endl;

}

else if (flag == 2)

cout << "\tacc" << endl;

else

error();

}

int getPos(char ch, bool isVT = true)

{

string::size\_type res;

if (isVT)

res = m\_VT.find(ch);

else

res = m\_VN.find(ch);

if (res == string::npos)

{

sprintf(m\_errorMsg, "The input symbol string contains illegal characters!");

error();

}

return res;

}

void shift(int stateNum)

{

char symbol = m\_target[m\_curPos++];

m\_stateStack.push\_back(stateNum);

m\_symbolStack.push\_back(Node(symbol));

}

void reduction(int stateNum)

{

string production = m\_production[stateNum - 1];

int len = production.size() - 3;

vector<Node> tempArr;

for (int i = m\_symbolStack.size() - len; i < m\_symbolStack.size(); i++)

{

tempArr.push\_back(m\_symbolStack[i]);

}

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

{

m\_stateStack.pop\_back();

m\_symbolStack.pop\_back();

}

Node ne = makeTAC(stateNum, tempArr);

m\_symbolStack.push\_back(ne);

int pos = getPos(production[0], false);

int topState = m\_stateStack.back();

int newState = m\_gotoTable[topState][pos];

// goto不存在

if (newState == 0)

sprintf(m\_errorMsg, "Goto(%d, %c) is null!", topState, production[0]);

m\_stateStack.push\_back(newState);

}

int getNewNum()

{

static int cur = 0;

return ++cur;

}

Node makeTAC(int stateNum, vector<Node> &arr)

{

if (6 == stateNum)

return Node('F', "i");

if (5 == stateNum)

return Node('F', arr[1].addr);

if (4 == stateNum)

return Node('T', arr[0].addr);

if (3 == stateNum || 1 == stateNum)

{

int num = getNewNum();

string sy = "a";

sy[0] = arr[1].symbol;

string neAddr = "t" + to\_string(num);

// 生成四元式

string tac = "(" + sy + "," + arr[0].addr + "," + arr[2].addr + "," + neAddr + ")";

m\_tac.push\_back(tac);

if (1 == stateNum)

return Node('E', neAddr);

return Node('T', neAddr);

}

if (2 == stateNum)

return Node('E', arr[0].addr);

return Node('#');

}

public:

void analyse(string &s)

{

init(s);

while (m\_curPos < m\_target.size())

{

int t = getPos(m\_target[m\_curPos]);

int i = m\_stateStack.back();

if (m\_actionTable[i][t] == "acc")

{

printStep(i, 2);

break;

}

// action不存在

else if (m\_actionTable[i][t] == "0")

{

sprintf(m\_errorMsg, "Action(%d, %d) is null!", i, t);

printStep(i, 3);

break;

}

else if (m\_actionTable[i][t][0] == 'S')

{

int stateNum = 0;

for (int j = 1; j < m\_actionTable[i][t].size(); j++)

stateNum = stateNum \* 10 + m\_actionTable[i][t][j] - '0';

printStep(stateNum, 0);

shift(stateNum);

}

else

{

int stateNum = 0;

for (int j = 1; j < m\_actionTable[i][t].size(); j++)

stateNum = stateNum \* 10 + m\_actionTable[i][t][j] - '0';

// 归约过程放在打印中,因为需要输出Goto值和操作前的栈状态

printStep(stateNum, 1);

}

}

cout << endl

<< "四元式如下:" << endl;

for (auto s : m\_tac)

cout << s << endl;

}

};

int main()

{

string s;

SLR1 analyer;

cout << endl

<< "SLR(1)分析程序并输出四元式,编制人:刘鑫,201958508239,计198-2" << endl;

cout << "输入一个以#结束的符号串(包括+\*/()i#):" << endl;

cin >> s;

cout << "输出过程如下:" << endl;

cout << "步骤"

<< "\t"

<< "状态栈"

<< "\t\t"

<< "符号栈"

<< "\t\t"

<< "剩余输入串"

<< "\t"

<< "Action"

<< "\t"

<< "Goto" << '\t'

<< "四元式"

<< endl;

analyer.analyse(s);

return 0;

}