### **第二次实验报告**

### 本次实验亮点部分：

第二次实验分别用4种方式（递归子程序下降法、LL(1)分析法、LR(0)分析法和算符优先分析法）实现语义分析；

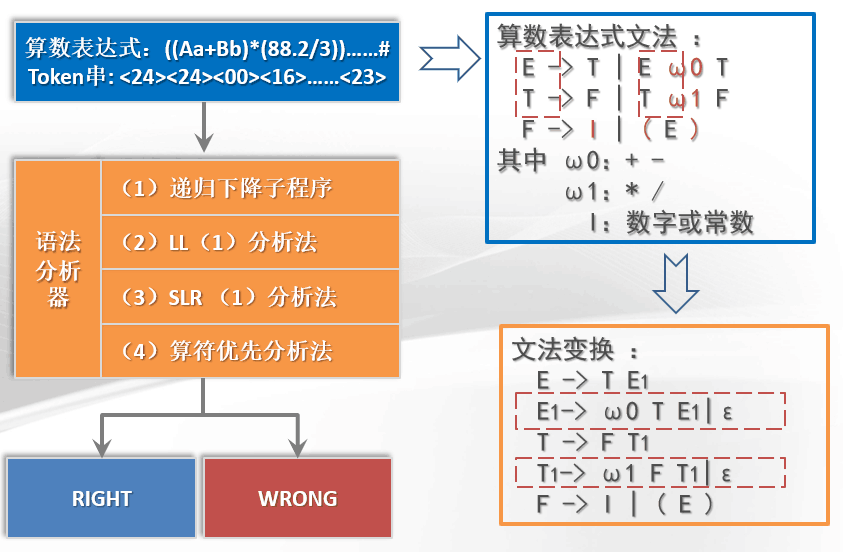
### **实验目标：**

表达式语法分析器的设计实现

### **实验内容：**

（包括：概要设计、数据结构、流程图、关键函数等有选择填写）

首先，利用第一次实验的结果，即token串，进行切词，作为输入；然后利用四种不同的文法分析方式进行文法分析，具体有：递归子程序下降法、LL(1)分析法、LR(0)分析法、算符优先分析法；分别进行了实现，并用以下示例进行验证：



### 2.1 Token输入格式调整

为了后续实验方便，每一个文法分析器的输入都是一个一维数组，这个数组就是从文件读取代码后所形成的TOKEN串，例如：

113,201,401,140,402,403,407...

代表：

int main(){}

调整后的提供TOKEN串的文件如下：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FileName:WordFilter1.h

Function: Provide token strings to grammar parsers from file.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <fstream>

#include <string>

#include <cassert>

using namespace std;

void wordFilter(int\* tokens)

{

char ch; //当前读入的字符

string str; //单词单位

int token; //TOKEN格式详见文件token.xlsx

string fileName = "a0.txt"; //从文件读入

ifstream infile;

infile.open(fileName.data());

assert(infile.is\_open());

infile >> noskipws;

infile >> ch;

//分别记录标识符、常量和界符，如果后续出现重复的词语则不再新增token，只与旧的token关联

string identifier[1024], constant[1024], boundary[1024];

int idNum = 0, conNum = 0, bounNum = 0;

while(!infile.eof() && ch!=EOF)

{

//过滤换行、Tab或空格

if(ch=='\n'||ch=='\t'||ch==' ')

{

infile >> ch;

}

//若是字母或下划线开头,是变量

else if((ch>='a' && ch<='z') ||

(ch>='A' && ch<='Z') ||

(ch == '\_'))

{

str = "";

token = 0;

while((

(ch>='a' && ch<='z') ||

(ch>='A' && ch<='Z') ||

(ch == '\_') ||

(ch>='0'&&ch<='9')) &&

(!infile.eof()) )

{

str = str + ch;

infile >> ch;

}

cout<<"("<<str<<",";

int haveId = 0;

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

{

if(identifier[i] == str)

{

token = 200 + i;

cout<<token<<")"<<endl;

tokens[tokenNumNow++] = token;

haveId = 1;

break;

}

}

if(haveId == 0)

{

identifier[idNum++] = str;

token = 200 + idNum;

cout<<token<<")"<<endl;

tokens[tokenNumNow++] = token;

}

}

//若是数字开头，是数字常量

else if(ch>='0' && ch<='9')

{

//token = 3;

str = "";

while(((ch>='0' && ch<='9')||(ch == '.')) && !(ch=='\n'||ch=='\t'||ch==' ') && (!infile.eof()) )

{

str = str + ch;

infile >> ch;

}

constant[conNum++] = str;

token = 300 + conNum;

cout<<"("<<str<<","<<token<<")"<<endl;

tokens[tokenNumNow++] = token;

}

//若是引号开头，是字符(串)常量

else if(ch == '\'' || ch == '\"')

{

// token = 3;

str = "";

str = str + ch;

infile >> ch;

while(ch != '\'' && ch != '\"' && !infile.eof())

{

str = str + ch;

infile >> ch;

}

str = str + ch;

infile >> ch;

constant[conNum++] = str;

token = 300 + conNum;

cout<<"("<<str<<","<<token<<")"<<endl;

tokens[tokenNumNow++] = token;

}

//+-\*/()#，是运算符(界符)

else if(ch == '+'||ch == '-')

{

str = "";

str = str + ch;

infile >> ch;

token = 401;

cout<<"("<<str<<","<<token<<")"<<endl;

tokens[tokenNumNow++] = token;

}

else if(ch == '\*'||ch == '/')

{

str = "";

str = str + ch;

infile >> ch;

token = 402;

cout<<"("<<str<<","<<token<<")"<<endl;

tokens[tokenNumNow++] = token;

}

else if(ch == '(')

{

str = "";

str = str + ch;

infile >> ch;

token = 403;

cout<<"("<<str<<","<<token<<")"<<endl;

tokens[tokenNumNow++] = token;

}

else if(ch == ')')

{

str = "";

str = str + ch;

infile >> ch;

token = 404;

cout<<"("<<str<<","<<token<<")"<<endl;

tokens[tokenNumNow++] = token;

}

else if(ch == '#')

{

str = "";

str = str + ch;

infile >> ch;

token = 405;

cout<<"("<<str<<","<<token<<")"<<endl;

tokens[tokenNumNow++] = token;

}

else

{

cout<<"ERROR!"<<endl;

tokens[tokenNumNow++] = 999;

infile >> ch;

}

}

infile.close();

/\*

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

{

cout<<"token"<<i<<":"<<tokens[i]<<endl;

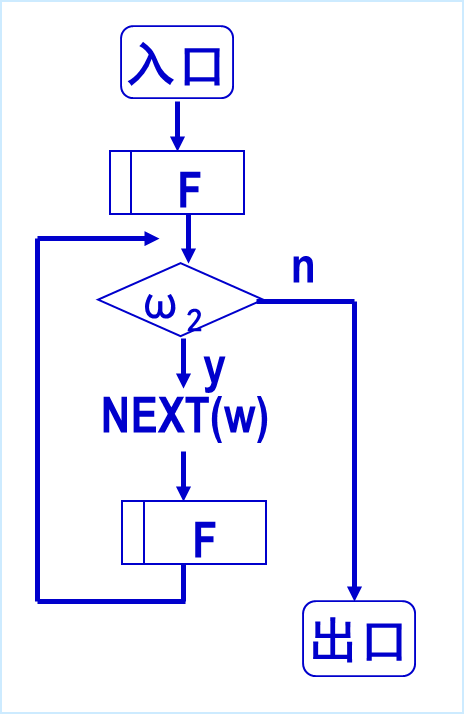
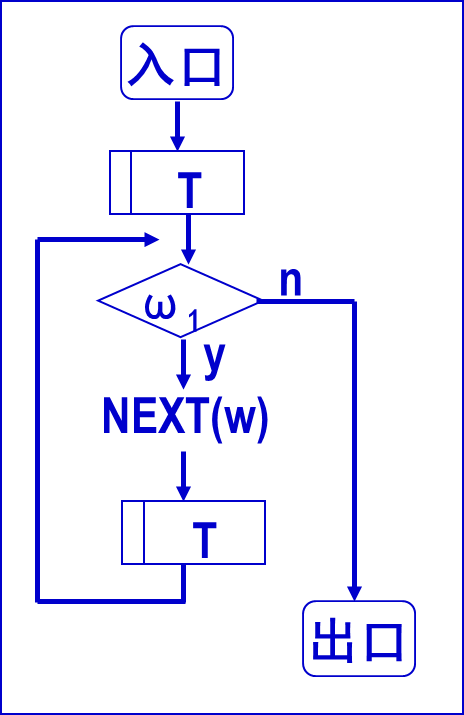
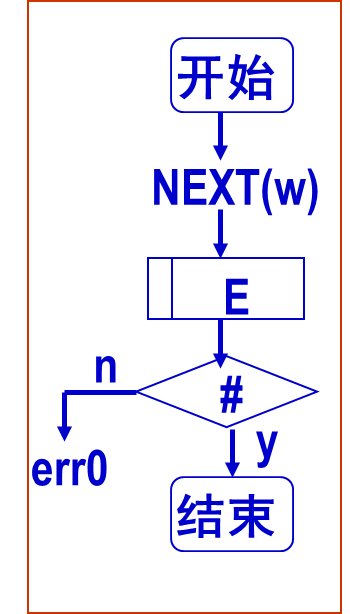
}

\*/

}

### **2.2 递归子程序下降法**

程序流程图如下所示：



主程序 子程序T 子程序E

源程序代码：（加入注释）

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FileName:ParserRD.cpp

Function: A recursive subprogram descent method is used for

grammar analysis.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <string>

#include "WordFilter1.h"

using namespace std;

int tokens[1024], tokenNumNow = 0;

void functionE();

void functionT();

void functionF();

bool flag = 1;

int main()

{

//生成TOKEN串并保存在全局变量tokens中

wordFilter(tokens);

//进入子程序E

functionE();

//如果没有发生错误并且最后token只剩下#

if(flag == 1 && tokens[tokenNumNow] == 405)

cout<<"Successful!"<<endl;

else

cout<<"Error!"<<endl;

return 0;

}

void functionE()

{

//进入子程序T

functionT();

//如果从T出来后，紧接着后面字符是+或者-

while(tokens[tokenNumNow]==401)

{

cout<<tokens[tokenNumNow]<<endl;

tokenNumNow++;

//继续进入子程序T

functionT();

}

}

void functionT()

{

//进入子程序T

functionF();

//如果从T出来后，紧接着后面字符是\*或者/

while(tokens[tokenNumNow] == 402)

{

cout<<tokens[tokenNumNow]<<endl;

tokenNumNow++;

//继续进入子程序T

functionF();

}

}

void functionF()

{

int flagF1=0,flagF2=0;

//如果当前字符是一个变量或者常量I

if(tokens[tokenNumNow]>=200 && tokens[tokenNumNow]<=399)

{

flagF1=1;

cout<<tokens[tokenNumNow]<<endl;

tokenNumNow++;

}

//若当前字符是{

else if(tokens[tokenNumNow]==403)

{

cout<<tokens[tokenNumNow]<<endl;

tokenNumNow++;

//进入子程序E

functionE();

//从T出来后，紧接着后面字符是否是}

if(tokens[tokenNumNow]==404)

{

flagF2 = 1;

cout<<tokens[tokenNumNow]<<endl;

tokenNumNow++;

}

}

else

{

flag = 0;

}

//进入F后字符要么是变量/常量要么是{，否则出错

if(!flagF1 && !flagF2)

{

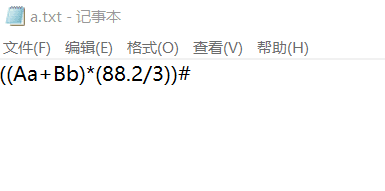
flag = 0;

}

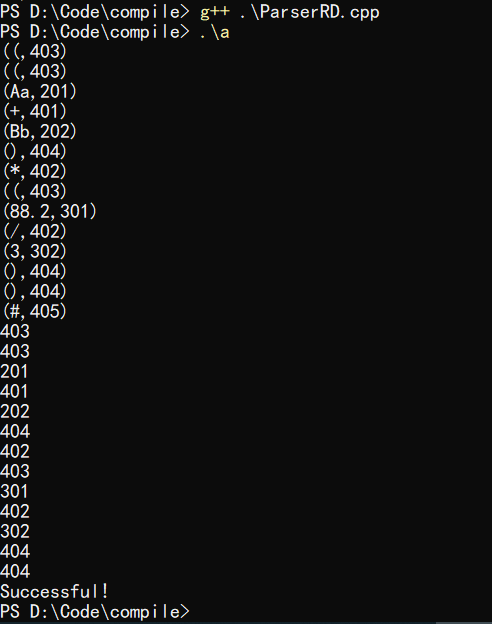
}

程序运行结果：（截屏）

输入：

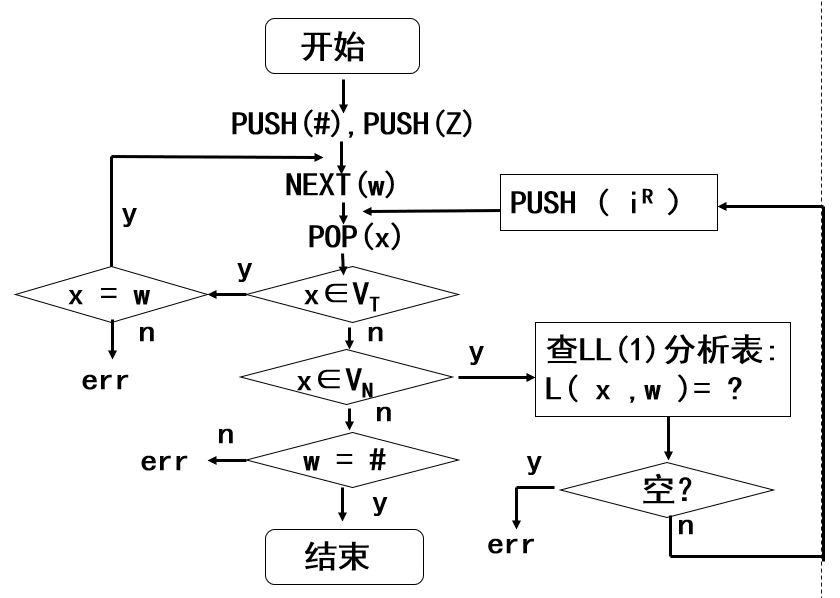


输出：



### **2.3 LL(1)分析法**

LL(1)分析法的流程图大致如下：



LL(1)分析表如下：



源程序代码：（加入注释）

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FileName:ParserLL.cpp

Function: LL (1) analysis method is used for grammar analysis.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <stack>

#include "WordFilter1.h"

using namespace std;

void reverseStack(int order);

int lookupTable(char x, int w);

bool endIsRight(char x, int w);

stack<char> s;

int tokens[1024], tokenNumNow = 0;

bool flag = 1;

//产生式逆序压栈

char E[2] = {'T','R'};

char R[3] = {'W','T','R'};

char T[2] = {'F','Y'};

char Y[3] = {'M','F','Y'};

char F[3] = {'(','E',')'};

//终结符表

char endSymbol[6] = {'I','W','M','(',')','#'};

//LL（1）分析表

int analysisTable[5][6] =

{

1, 0, 0, 1, 0, 0,

0, 2, 0, 0, 3, 3,

4, 0, 0, 4, 0, 0,

0, 6, 5, 0, 6, 6,

7, 0, 0, 8, 0, 0

};

int main()

{

char x;

int w;

wordFilter(tokens);

s.push('#');

s.push('E');

while(x != '#')

{

x = s.top();

s.pop();

w = tokens[tokenNumNow];

cout<<"x="<<x<<","<<"w="<<w<<endl;

bool xIsEndSymbol = 0;

for(int i = 0; i < sizeof(endSymbol);i++)

{

//若栈顶符号是终结符，当前token移进

if (x == endSymbol[i])

{

xIsEndSymbol = 1;

if(endIsRight(x,w)) {tokenNumNow++; break;}

else {flag = 0;break;}

}

}

//若x不是中介符号，逆序压栈

if(xIsEndSymbol == 0)

{

int orderhere = lookupTable(x,w);

if(orderhere == 0){flag = 0; break;}

else{reverseStack(orderhere);}

}

}

if(flag == 1 && x == '#' && w == 405)

cout<<"Successful!"<<endl;

else

cout<<"Error!"<<endl;

return 0;

}

//返回选择的产生式序号

int lookupTable(char x, int w)

{

int row, column;

switch(x)

{

case 'E':{row = 0;break;}

case 'R':{row = 1;break;}

case 'T':{row = 2;break;}

case 'Y':{row = 3;break;}

case 'F':{row = 4;break;}

default:{cout<<"ERROR::at LookupTable();"<<endl;break;}

}

switch(w)

{

case 401:{column = 1;break;}

case 402:{column = 2;break;}

case 403:{column = 3;break;}

case 404:{column = 4;break;}

case 405:{column = 5;break;}

default:{column = 0;break;}

}

cout<<"choose:"<<analysisTable[row][column]<<endl;

return analysisTable[row][column];

}

//按照产生式序号进行逆序压栈

void reverseStack(int order)

{

switch(order)

{

case 1:{for(int i = sizeof(E)-1; i>=0;s.push(E[i]), i--);break;}

case 2:{for(int i = sizeof(R)-1; i>=0;s.push(R[i]), i--);break;}

case 4:{for(int i = sizeof(T)-1; i>=0;s.push(T[i]), i--);break;}

case 5:{for(int i = sizeof(Y)-1; i>=0;s.push(Y[i]), i--);break;}

case 8:{for(int i = sizeof(F)-1; i>=0;s.push(F[i]), i--);break;}

case 3:{break;}

case 6:{break;}

case 7:{s.push('I');break;}

default:{cout<<"ERROR!"<<endl;break;}

}

}

//判断栈顶符号（终结符）与当前token是否对应

bool endIsRight(char x, int w)

{

switch(x)

{

case 'I':{if(w >= 200 && w < 399) return 1;else return 0;break;}

case 'W':{if(w == 401) return 1;else return 0;break;}

case 'M':{if(w == 402) return 1;else return 0;break;}

case '(':{if(w == 403) return 1;else return 0;break;}

case ')':{if(w == 404) return 1;else return 0;break;}

case '#':{if(w == 405) return 1;else return 0;break;}

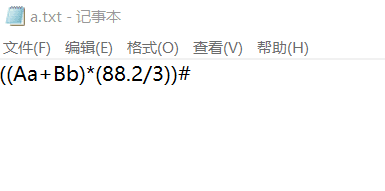
default:{cout<<"ERROR::at endIsRight();"<<endl;return 0;break;}

}

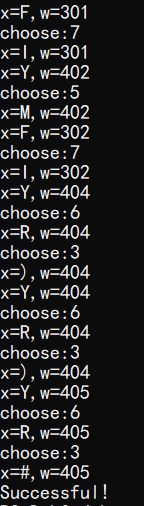
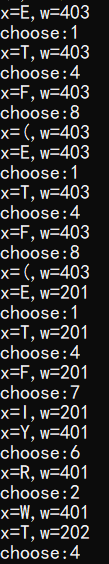
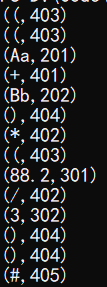
}

程序运行结果：（截屏）

输入：



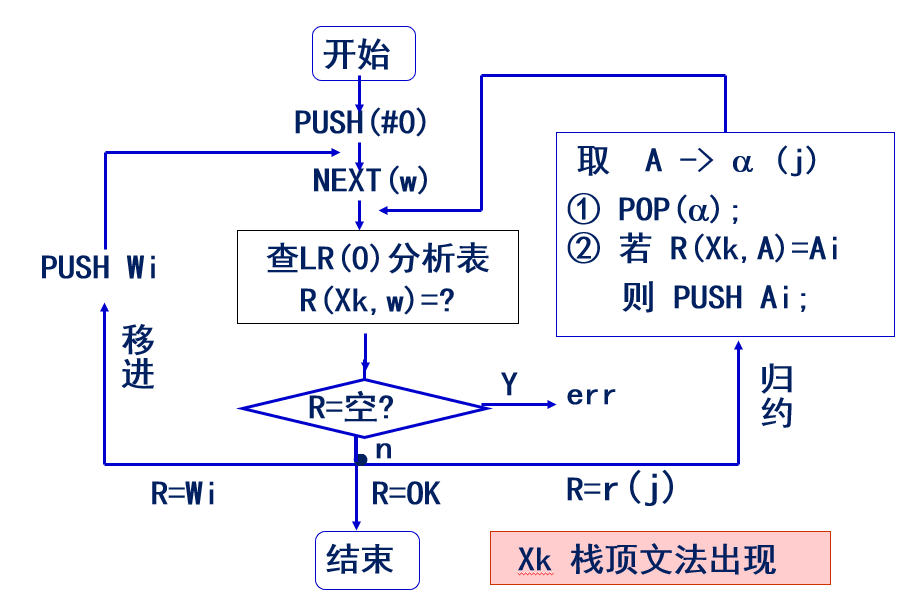
输出：



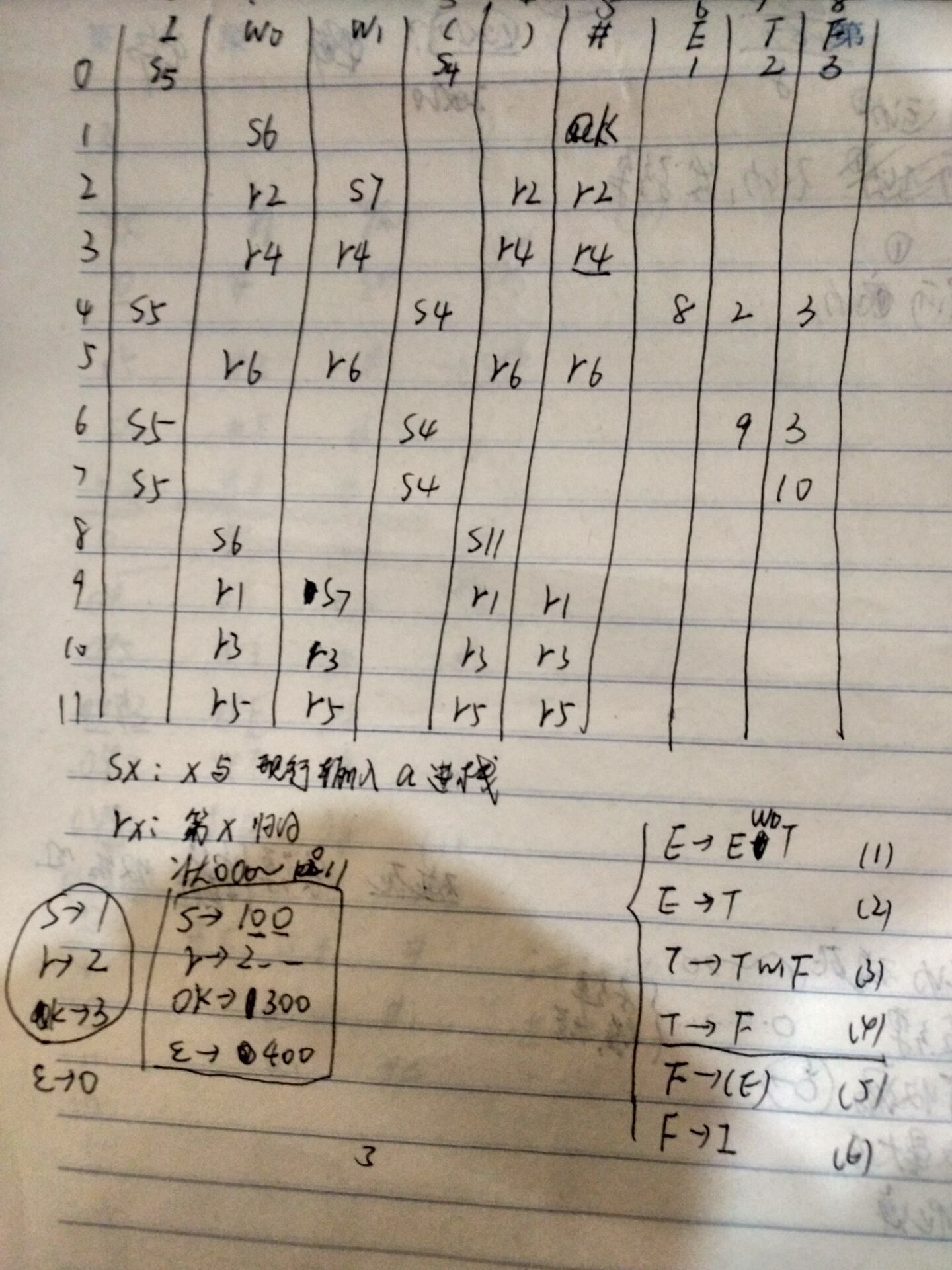
（其中，x是栈顶符号，w是当前token符号，choose代表选择的产生式）

### **2.4 LR(0)分析法**

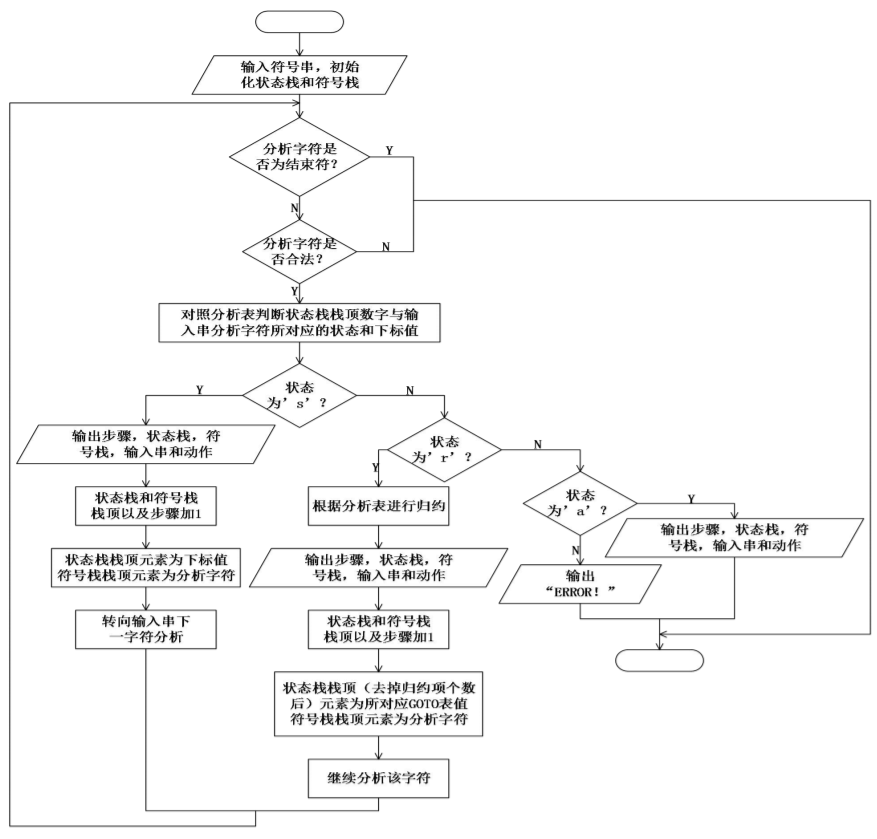
流程图大致如下：



LR（0）分析表如下：



其中， Sx指把下一状态x和现行输入符号w移进栈；Rx指按文法的第x个产 生式进行归约；OK表示分析成功；空白格为出错。在程序中，状态采用0~11表示，Sx使用101~111表示，Rx使用201~207表示，OK使用300表示，空格使用400表示。这样更容易直接用数组和数字来判断状态。具体程序流程图如下：



源程序代码：（加入注释）

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FileName:ParserLR0.cpp

Function: LR (0) analysis is used for grammar analysis.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <stack>

#include "WordFilter1.h"

using namespace std;

//LR(0)分析表

int analysisTable[12][9] =

{

// I W0 W1 ( ) # E T F

105,400,400,104,400,400, 1, 2, 3,

400,106,400,400,400,300,400,400,400,

400,202,107,400,202,202,400,400,400,

400,204,204,400,204,204,400,400,400,

105,400,400,104,400,400, 8, 2, 3,

400,206,206,400,206,206,400,400,400,

105,400,400,104,400,400,400, 9, 3,

105,400,400,104,400,400,400,400, 10,

400,106,400,400,111,400,400,400,400,

400,201,107,400,201,201,400,400,400,

400,203,203,400,203,203,400,400,400,

400,205,205,400,205,205,400,400,400

};

int getPlusCode(int x, int w);

int getReduceCode(int x, char c);

char getSymbol(int w);

void reduceSymbolStack(int order);

int tokens[1024], tokenNumNow = 0, flag = 2;

stack<int> statusStack;

stack<char> symbolStack;

int main()

{

wordFilter(tokens);

statusStack.push(0);

symbolStack.push('#');

int x, w, code1, code2;

char c;

while(flag!=0 && flag!=1)

{

x = statusStack.top();

w = tokens[tokenNumNow];

c = symbolStack.top();

//code1得到LR(0)分析表中Sx或Rx的值

code1 = getPlusCode(x,w);

cout<<"code1="<<code1<<endl;

if(code1 == 400)

flag = 0;

else if(code1 == 300)

flag = 1;

//移进

else if(code1>100 && code1<200)

{

statusStack.push(code1 - 100);

symbolStack.push(getSymbol(w));

tokenNumNow++;

}

//归约

else if(code1>200 && code1<300)

{

reduceSymbolStack(code1 - 200);

c = symbolStack.top();

x = statusStack.top();

//code2是状态转换序号

code2 = getReduceCode(x,c);

cout<<"code2="<<code2<<endl;

if(code2 > 10 || code2 < 0)

flag = 0;

else

{

statusStack.push(code2);

}

}

else

{

flag = 0;

}

}

if(flag == 1)

cout<<"Successful"<<endl;

else

cout<<"Error"<<endl;

return 0;

}

//获取移进或归约序号

int getPlusCode(int x, int w)

{

int row, column;

row = x;

switch(w)

{

case 401:{column = 1;break;}

case 402:{column = 2;break;}

case 403:{column = 3;break;}

case 404:{column = 4;break;}

case 405:{column = 5;break;}

default:{column = 0;break;}

}

return analysisTable[row][column];

}

//获取状态转换序号

int getReduceCode(int x, char c)

{

int row, column;

row = x;

switch(c)

{

case 'E':{column = 6;break;}

case 'T':{column = 7;break;}

case 'F':{column = 8;break;}

default:{cout<<"ERROR::at getReduceCode();"<<endl;flag = 0;break;}

}

return analysisTable[row][column];

}

//获取token对应的分析表符号

char getSymbol(int w)

{

char res;

switch(w)

{

case 401:{res = 'w';break;}

case 402:{res = 'm';break;}

case 403:{res = '(';break;}

case 404:{res = ')';break;}

case 405:{res = '#';break;}

default:{res = 'i';break;}

}

return res;

}

//归约动作

void reduceSymbolStack(int order)

{

switch(order)

{

case 1:{symbolStack.pop();symbolStack.pop();statusStack.pop();statusStack.pop();statusStack.pop();break;}

case 2:{symbolStack.pop();symbolStack.push('E');statusStack.pop();break;}

case 3:{symbolStack.pop();symbolStack.pop();statusStack.pop();statusStack.pop();statusStack.pop();break;}

case 4:{symbolStack.pop();symbolStack.push('T');statusStack.pop();break;}

case 5:{symbolStack.pop();symbolStack.pop();symbolStack.pop();symbolStack.push('F');statusStack.pop();statusStack.pop();statusStack.pop();break;}

case 6:{symbolStack.pop();symbolStack.push('F');statusStack.pop();break;}

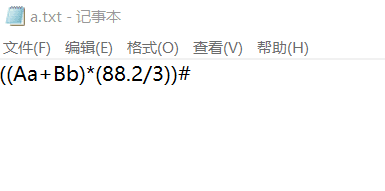
default:{cout<<"ERROR::at reduceSymbolStack()"<<endl;flag = 0;break;}

}

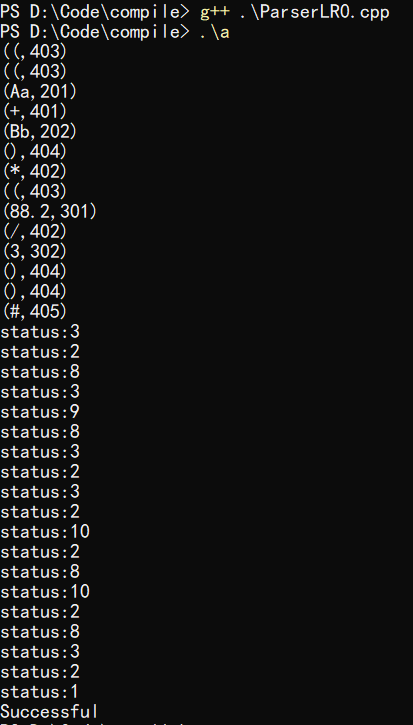
}

程序运行结果：（截屏）

输入：

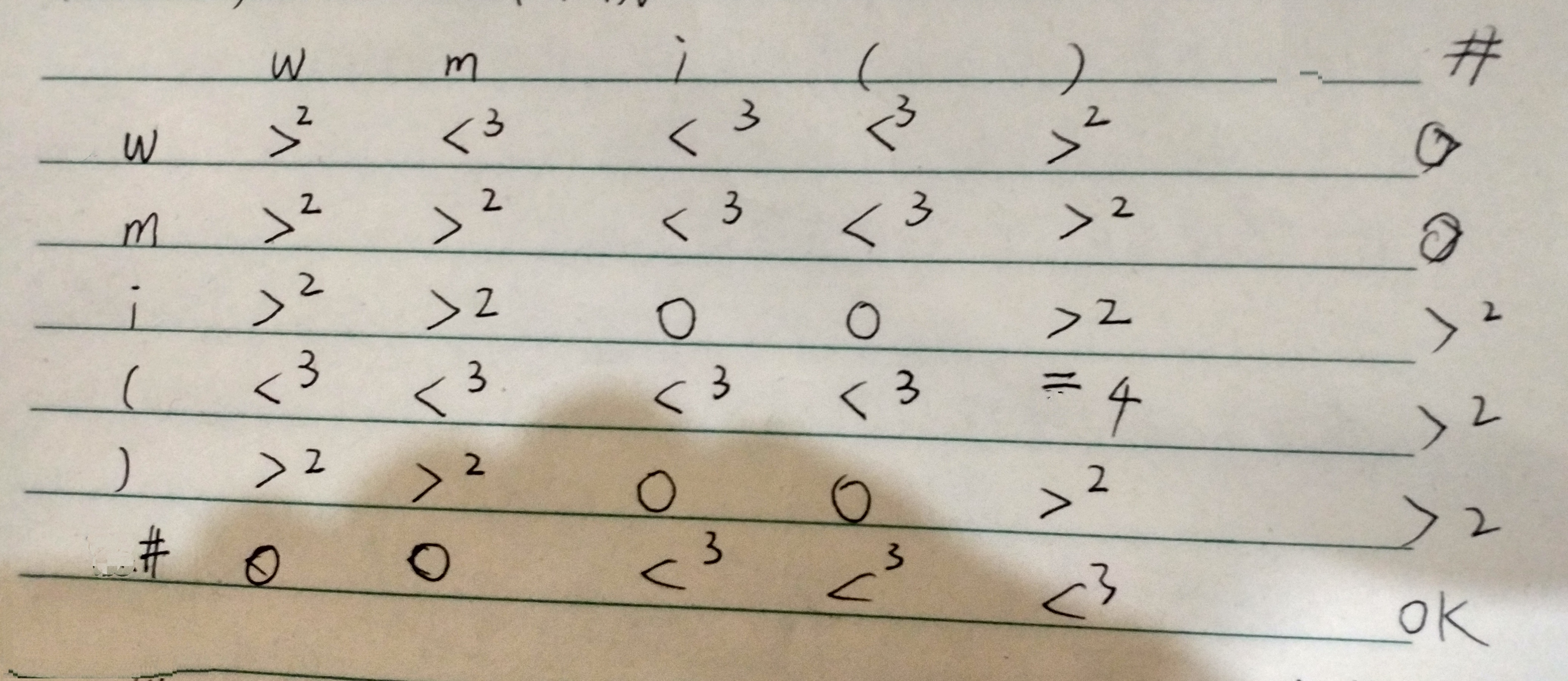


输出：



2.4 算符优先分析法

算符优先关系表如下：



在程序中，空格为0，即失败，OK为1，>为2，<为3，=为4。

源程序代码：（加入注释）

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FileName:ParserSF.cpp

Function: Operator-first method is used for grammar analysis.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <stack>

#include "WordFilter1.h"

using namespace std;

//算符优先关系表

int relationTable[6][6] =

{

// +,\*,i,(,),#

2,3,3,3,2,0,

2,2,3,3,2,0,

2,2,0,0,2,2,

3,3,3,3,4,2,

2,2,0,0,2,2,

0,0,3,3,3,1

};//Error:0,OK:1,>:2,<:3,=:4

int getRelation(char x, int w);

char getSymbol(int w);

int tokens[1024], tokenNumNow = 0;

stack<char> symbolStack;

int flag = 2;

int main()

{

char x;

int w;

wordFilter(tokens);

symbolStack.push('#');

x = symbolStack.top();

w = tokens[tokenNumNow];

while(flag!=0 && flag!=1)

{

x = symbolStack.top();

w = tokens[tokenNumNow];

if(x!='V')

{

int relation = getRelation(x,w);

cout<<"x="<<x<<",w="<<w<<",r="<<relation<<",";

if(relation==3 || relation==4) //移进

{

cout<<"Move In."<<endl;

symbolStack.push(getSymbol(w));

tokenNumNow++;

}

else if(relation==0 || relation==1)

{

flag = relation;

}

else if(relation == 2) //归约

{

cout<<"Reduction."<<endl;

switch(x)

{

case 'i':{symbolStack.pop();symbolStack.push('V');break;}

case ')':{symbolStack.pop();symbolStack.pop();symbolStack.pop();symbolStack.push('V');break;}

default:{flag = 0;cout<<"War::at Guiyue;"<<endl;break;}

}

}

}

else

{

cout<<"x="<<x<<",w="<<w<<",r=";

symbolStack.pop();

x = symbolStack.top();

if(x == 'V') {flag = 0;cout<<"x2=V"<<endl;break;}

int relation = getRelation(x,w);

cout<<relation<<",";

if(relation==3 || relation==4) //移进

{

cout<<"Move In."<<endl;

symbolStack.push('V');

symbolStack.push(getSymbol(w));

tokenNumNow++;

}

else if(relation==0 || relation==1)

{

flag = relation;

}

else if(relation == 2) //归约

{

cout<<"Reduction."<<endl;

if(x=='w' ||x=='m')

{

symbolStack.pop();

}

else

{

cout<<"?V,?!=+-\*/"<<endl;

flag = 0;

}

}

}

}

x = symbolStack.top();

if(flag == 1 && w == 405 && x == '#')

cout<<"Successful!"<<endl;

else

cout<<"Error!"<<endl;

return 0;

}

//获取栈顶符号和当前输入符号的关系

int getRelation(char x, int w)

{

int row, column;

switch(x)

{

case 'w':{row = 0;break;}

case 'm':{row = 1;break;}

case 'i':{row = 2;break;}

case '(':{row = 3;break;}

case ')':{row = 4;break;}

case '#':{row = 5;break;}

default:{cout<<"Err::at getRelation();"<<endl;break;}

}

switch(w)

{

case 401:{column = 0;break;}

case 402:{column = 1;break;}

case 403:{column = 3;break;}

case 404:{column = 4;break;}

case 405:{column = 5;break;}

default:{column = 2;break;}

}

return relationTable[row][column];

}

//获取token对应的符号

char getSymbol(int w)

{

char symbol;

switch(w)

{

case 401:{symbol = 'w';break;}

case 402:{symbol = 'm';break;}

case 403:{symbol = '(';break;}

case 404:{symbol = ')';break;}

case 405:{symbol = '#';break;}

default:{symbol = 'i';break;}

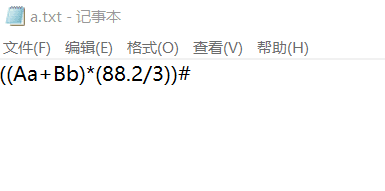
}

return symbol;

}

程序运行结果：（截屏）

输入：



输出：

