# 程序设计2：LL1

曹桢 学号：2014211182 班级：2014211302

## 实验题目：

语法分析程序的设计与实现。

## 实验内容：

编写语法分析程序，实现对算术表达式的语法分析。产生式如下：

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

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

F-> id | (E) | num

实验要求：在对输入的算数表达式进行分析的过程中，依次输出所采用的产生式。编写LL(1)语法分析程序，要求如下：

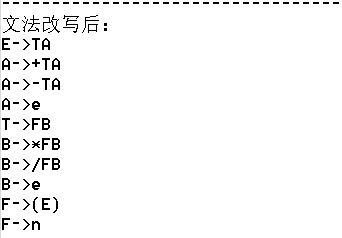
1. 编程实现算法4.2，为给定文法自动构造预测分析表。
2. 编程实现算法4.1，构造LL(1)预测分析程序。并进行错误处理。

## 实现要求：

采用C/C++作为实现语言，手工编写语法分析程序。

## 程序设计说明

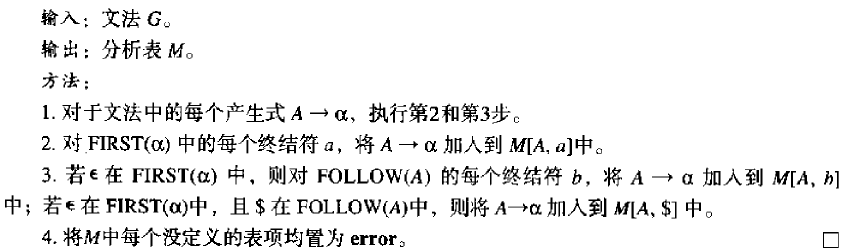
1. 输入文件：待分析字符串
2. 输出：对符号串的分析过程
3. 算法思想：
4. 改写文法，消除左递归

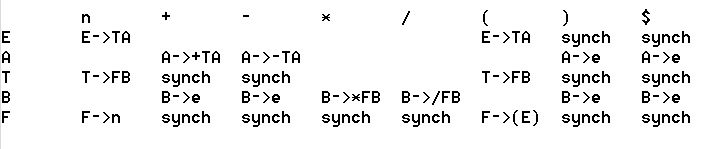


1. 构造First集和Follow集

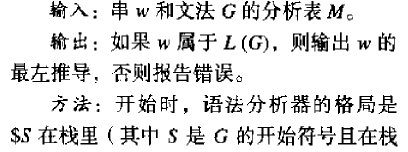


1. 算法4.2，构造分析表

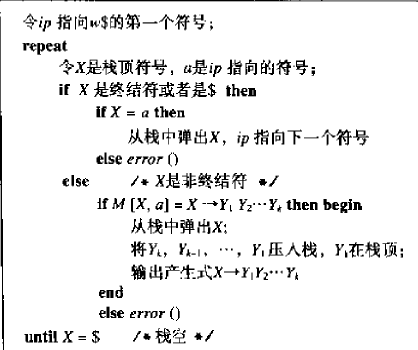




1. 算法4.1，LL(1)分析







1. 高层数据结构设计
2. 常量定义

const int Nsize = 5; //非终结符个数

const int Vsize = 8; //终结符个数

const int Gsize = 10; //产生式个数

1. 关键变量定义

char N[Nsize]={'E','A','T','B','F'}; //非终结符

char T[Vsize]={'n','+','-','\*','/','(',')','$'}; //终结符

vector<char> First[Nsize]; //First集

vector<char> Follow[Nsize]; //Follow集

stack<char> AnalyseStack; //分析栈

string AnalyseTable[Nsize][Vsize]; //预测分析表

string generation[Gsize]; //消除左递归后的产生式

string inputBuffer; //输入缓冲区

int ip; //输入缓冲区指针

1. 函数说明

/\*把非终结符映射到数字\*/

int mapN(char c)

/\*把终结符映射到数字\*/

int mapV(char c)

/\*把输入字符串中的数变成n\*/

string numberToN(string w)

/\*构建分析表\*/

void createAnalyseTable()

/\*错误处理\*/

void error1()

/\*LL(1)语法分析\*/

void LL1(string w)

## 源代码

/\*coder：曹桢 最后修改时间：2013-10-30 0：18

实验：程序设计2 方法2 LL(1)语法分析程序

\*/

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <stack>

#include <vector>

#include<iomanip>

using namespace std;

const int Nsize = 5; //非终结符个数

const int Vsize = 8; //终结符个数

const int Gsize = 10; //产生式个数

char N[Nsize]={'E','A','T','B','F'}; //非终结符

char T[Vsize]={'n','+','-','\*','/','(',')','$'}; //终结符

vector<char> First[Nsize]; //First集

vector<char> Follow[Nsize]; //Follow集

stack<char> AnalyseStack; //分析栈

string AnalyseTable[Nsize][Vsize]; //预测分析表

string generation[Gsize]; //消除左递归后的产生式

string inputBuffer; //输入缓冲区

int ip; //输入缓冲区指针

void Init()

{

/\*First集\*/

First[0].push\_back('(');

First[0].push\_back('n');

First[1].push\_back('+');

First[1].push\_back('-');

First[1].push\_back('e');

First[2].push\_back('(');

First[2].push\_back('n');

First[3].push\_back('\*');

First[3].push\_back('/');

First[3].push\_back('e');

First[4].push\_back('(');

First[4].push\_back('n');

/\*Follow集\*/

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

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

Follow[1].push\_back('$');

Follow[1].push\_back(')');

Follow[2].push\_back('$');

Follow[2].push\_back(')');

Follow[2].push\_back('+');

Follow[2].push\_back('-');

Follow[3].push\_back('$');

Follow[3].push\_back(')');

Follow[3].push\_back('+');

Follow[3].push\_back('-');

Follow[4].push\_back('$');

Follow[4].push\_back(')');

Follow[4].push\_back('+');

Follow[4].push\_back('-');

Follow[4].push\_back('\*');

Follow[4].push\_back('/');

/\*消除左递归后的产生式\*/

generation[0] = "E->TA";

generation[1] = "A->+TA";

generation[2] = "A->-TA";

generation[3] = "A->e";

generation[4] = "T->FB";

generation[5] = "B->\*FB";

generation[6] = "B->/FB";

generation[7] = "B->e";

generation[8] = "F->(E)";

generation[9] = "F->n";

/\*预测分析表\*/

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

for(int j = 0; j < Vsize; j++){

AnalyseTable[i][j] = "";

}

}

}

/\*把非终结符映射到数字\*/

int mapN(char c)

{

switch(c){

case 'E':

return 0;

case 'A':

return 1;

case 'T':

return 2;

case 'B':

return 3;

case 'F':

return 4;

default:

return -1;

}

}

/\*把终结符映射到数字\*/

int mapV(char c)

{

switch(c){

case 'n':

return 0;

case '+':

return 1;

case '-':

return 2;

case '\*':

return 3;

case '/':

return 4;

case '(':

return 5;

case ')':

return 6;

case '$':

return 7;

default:

return -1;

}

}

/\*把输入字符串中的数变成n\*/

string numberToN(string w)

{

bool flag = false;

string result = "";

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

while(w[i] >= '0' && w[i] <= '9' && i < w.length()){

flag = true;

i++;

}

if(flag == true){

result = result + 'n';

}

if(i != w.length()){

result = result + w[i];

flag = false;

}

}

return result;

}

/\*构建分析表\*/

void createAnalyseTable()

{

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

char alpha = generation[i][3];

int head = mapN(generation[i][0]);

int al = -1;

switch(alpha){

case 'n':case '+':case '-':case '\*':case '/':case '(':case ')':case '$':

al = mapV(generation[i][3]);

AnalyseTable[head][al] = generation[i];

break;

case 'E':case 'A':case 'T':case 'B':case 'F':

al = mapN(generation[i][3]);

for(int j = 0; j < First[al].size(); j++){

if(First[al][j] != 'e'){

int index = mapV(First[al][j]);

AnalyseTable[head][index] = generation[i];

}else{

for(int k = 0; k < Follow[head].size(); k++){

int index = mapV(Follow[head][k]);

AnalyseTable[head][index] = generation[i];

}

}

}

break;

case 'e':

for(int k = 0; k < Follow[head].size(); k++){

int index = mapV(Follow[head][k]);

AnalyseTable[head][index] = generation[i];

}

break;

}

}

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

for(int j = 0; j < Vsize; j++){

if(AnalyseTable[i][j].empty()){

for(int k = 0; k < Follow[i].size(); k++){

if(j == mapV(Follow[i][k])){

AnalyseTable[i][j] = "synch";

}

}

}

}

}

}

/\*错误处理\*/

void error1()

{

AnalyseStack.pop();

}

/\*LL(1)语法分析\*/

void LL1(string w)

{

char X;

char a;

int step = 0;

ip = 0;

inputBuffer = w+"$";

bool result = true;

stack<char> temp;

AnalyseStack.push('$');

AnalyseStack.push('E');

cout<<left<<setw(10)<<"步骤"<<left<<setw(10)<<"栈顶元素"<<left<<setw(20)<<"输入"<<left<<setw(20)<<"输出"<<endl;

do{

X = AnalyseStack.top();

a = inputBuffer[ip];

step++;

cout<<left<<setw(10)<<step<<left<<setw(10)<<X<<left<<setw(20)<<inputBuffer.substr(ip);

if(X == 'n' || X == '+' || X == '-' || X == '\*' || X == '/' || X == '(' || X == ')' || X == '$'){

if(X == a){

char c;

AnalyseStack.pop();

ip++;

cout<<X<<"匹配"<<endl;

}else{

error1();

}

}else if(X == 'E' || X == 'A' || X == 'T' || X == 'B' || X == 'F'){

int row = mapN(X);

int column = mapV(a);

if(row == -1 || column == -1){

cout<<"输入中含有不属于文法的终结符或非终结符"<<endl;

exit(0);

}

if(AnalyseTable[row][column].empty()){

ip++;

result = false;

cout<<"Error"<<endl;

}else if(AnalyseTable[row][column] == "synch"){

error1();

result = false;

cout<<"Error"<<endl;

}else{

if(AnalyseTable[row][column][3] == 'e'){

AnalyseStack.pop();

cout<<AnalyseTable[row][column]<<endl;

}else{

AnalyseStack.pop();

for(int i = AnalyseTable[row][column].length()-1; i >=3 ; i--){

AnalyseStack.push(AnalyseTable[row][column][i]);

}

cout<<AnalyseTable[row][column]<<endl;

}

}

}

}while(X != '$');

if(result == true){

cout<<"分析结果：正确！"<<endl;

}else{

cout<<"分析结果：错误！"<<endl;

}

}

/\*输出\*/

void printout()

{

cout<<"由给定产生式进行语法分析，产生式如下："<<endl;

cout<<"E->E+T | E-T | T\nT->T\*F | T/F | F\nF-> (E) | num"<<endl;

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

cout<<"文法改写后："<<endl;

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

cout<<generation[i]<<endl;

}

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

cout<<"\tFirst\tFollow"<<endl;

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

cout<<N[i]<<"\t";

for(int j = 0; j < First[i].size(); j++){

cout<<First[i][j]<<" ";

}

cout<<"\t";

for(int j = 0; j < Follow[i].size(); j++){

cout<<Follow[i][j]<<" ";

}

cout<<"\n";

}

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

for(int j = 0; j < Vsize; j++){

cout<<"\t"<<T[j];

}

cout<<endl;

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

cout<<N[i]<<"\t";

for(int j = 0; j < Vsize; j++){

cout<<AnalyseTable[i][j]<<"\t";

}

cout<<endl;

}

cout<<endl;

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

}

int main()

{

string w;

string y;

cout<<"请输入要分析的符号串：";

cin>>w;

y = numberToN(w);

Init();

createAnalyseTable();

printout();

LL1(y);

return 0;

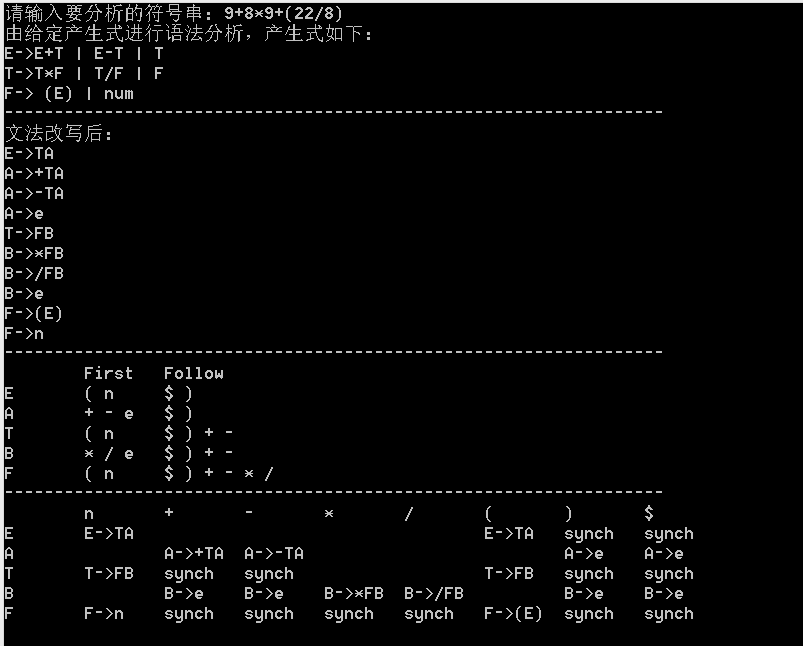
}

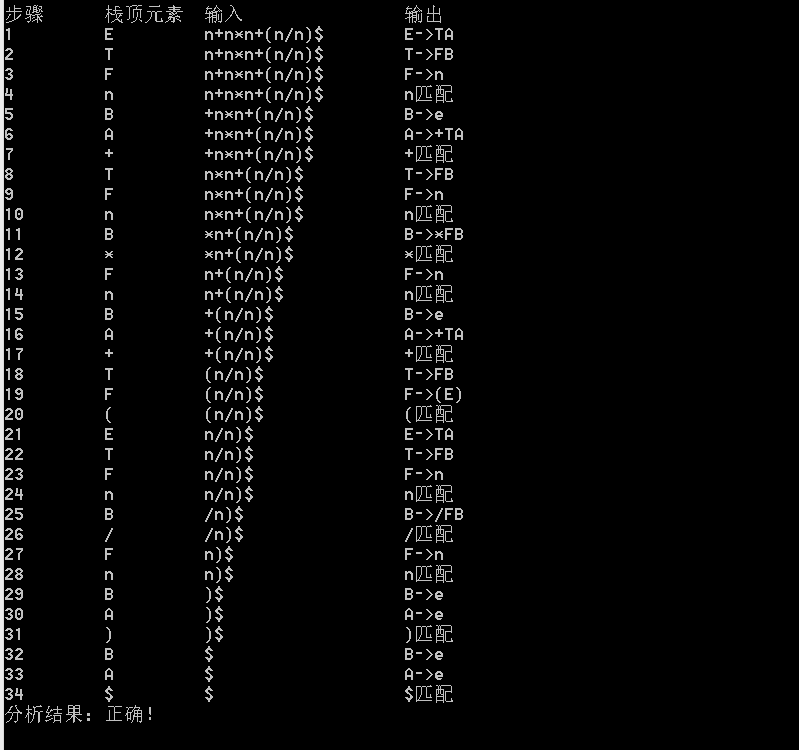
## 测试报告

1、

输入：9+8\*9+(22/8)

输出：



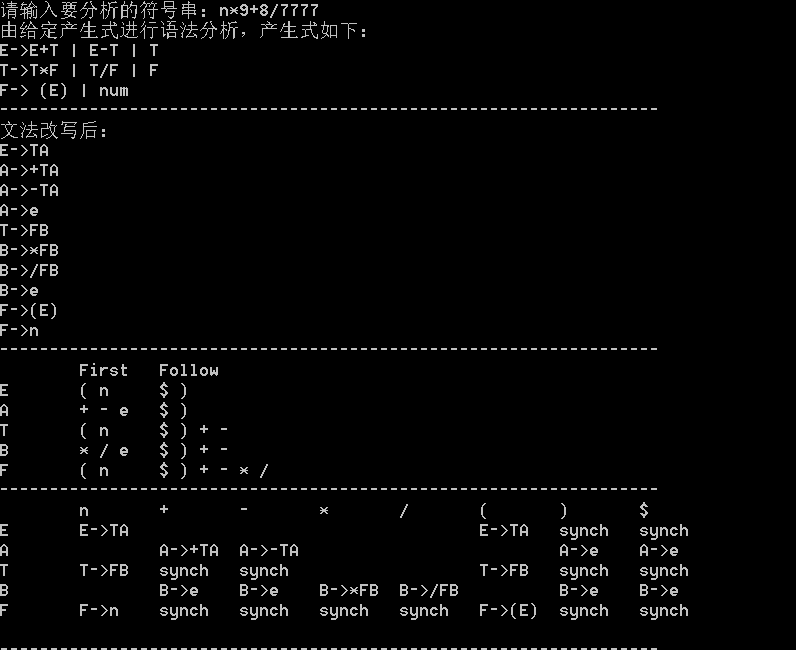


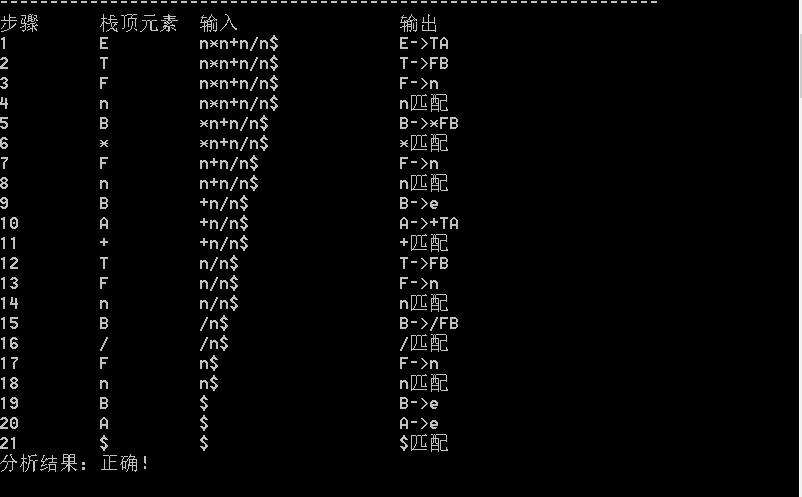
分析说明：能识别正确的符号串。

2、

输入：n\*9+8/7777

输出：



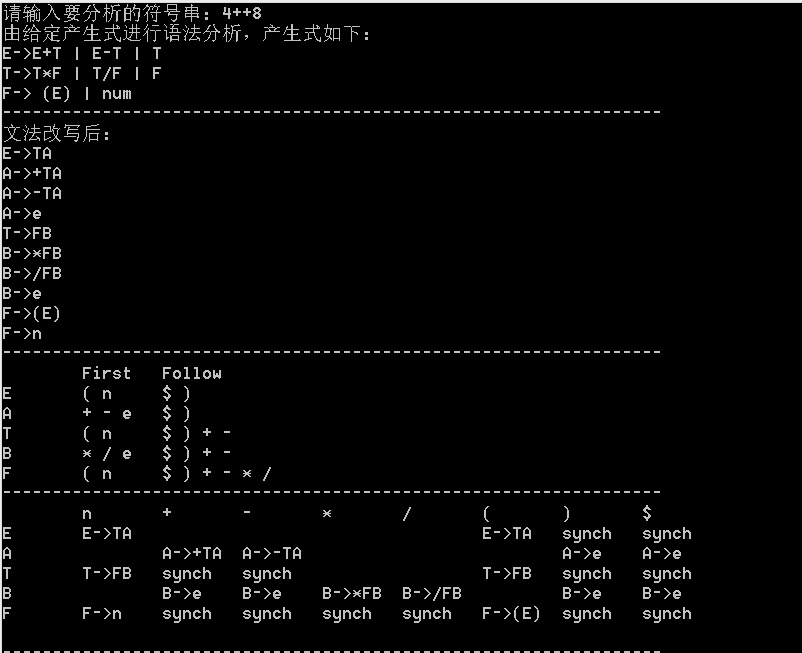


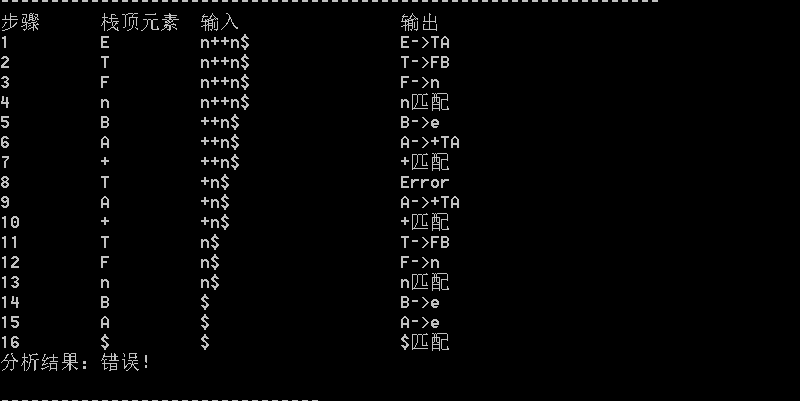
分析说明：能识别正确的符号串。

3、

输入：4++8

输出：





分析说明：能识别错误的符号串。