

编译原理与技术

LL(1)语法分析实验



2017-11-20

[裴子祥 计科七班 学号2015211921]

[指导老师：刘辰]

目录

[一．实验内容与要求 1](#_Toc499711854)

[二． 程序设计与实现 1](#_Toc499711855)

[1． 主要步骤 1](#_Toc499711856)

[2． 主要算法 1](#_Toc499711857)

[3． 程序实现 2](#_Toc499711858)

[4． 样例结果 4](#_Toc499711859)

[5． 实验总结 4](#_Toc499711860)

[6． 源码附件 6](#_Toc499711861)

# **一．实验内容与要求**

**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：

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。

**实验环境：**

**MICSOFT WINDOWS 10**

**Visual Studio 2015**

**C/C++作为实现语言**

1. 程序设计与实现
2. 主要步骤

(1) 消除文法的左递归

(2) 提取文法的左公因子

(3) 构造文法的FIRST集

(4) 构造文法的FOLLOW集

(5) 构造预测分析表

(6) 构造预测分析程序

1. 主要算法

**算法4.2**----------------------------->构造预测分析表

输入：文法G

输出：文法G的预测分析表M

方法：

for (文法G的每个产生式A->α) {

for (每个终结符号a∈FIRST(α))

把A->α放入M[A,a]中;

if (ε∈FIRST(α)

for (任何b∈OLLOW(A))

把A->阿尔法放入M[A,b]中;

}；

for (所有无定义的M[A,a]) 标上错误标志;

**算法4.1**----------------------------->构造预测分析程序

输入：输入符号串，文法G的一张预测分析表M。

输出：若在L(G)中，则输出的最左推导，否则报告错误。

方法：分析开始时，$在栈底，文法开始符号S在栈顶，$在输入缓冲区中

置ip指向 $ 的第一个符号；

do {

令X是栈顶符号，a是ip所指向的符号；

if (X是终结符号或$) {

if (X==a) {

从栈顶弹出X; ip前移一个位置;

};

else error();

else /\* X是非终结符号 \*/

if (M[X,a]=X->Y1Y2…Yk ) {

从栈顶弹出X;

把Yk、Yk-1、…、Y2、Y1压入栈，Y1在栈顶;

输出产生式XY1Y2…Yk ;

};

else error();

}while(X!=$) /\* 栈不空，继续 \*/

1. 程序实现
2. **输入**

待分析表达式字符串

如\*180\*62\*+3

10+(1+2)\*3+(22/8)

1. **输出文件形式部分举例**

对符号串的分析过程

1. **文法改写**

消除左递归：

E->TA

A->+TA | -TA | ε

T->FB

B->\*FB | /FB | ε

F->(E) | n

提取左公因式

没有左公因式可以提取

1. **FIRST集与FOLLOW集**

|  |  |  |
| --- | --- | --- |
|  | FIRST | FOLLOW |
| E | ( n | $ ) |
| A | + - ε | $ ) |
| T | ( n | $ + - ) |
| B | \* / ε | $ + - ) |
| F | ( n | $ \* / ) + - |

1. **全局变量**

const int VnSize = 5; //非终结符个数

const int VtSize = 8; //终结符个数

const int ProductionSize = 10; //产生式个数

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

char Vt[VtSize] = { 'n','+','-','\*','/','(',')','$' };//终结符

vector<char> FIRST[VnSize]; //FIRST集

vector<char> FOLLOW[VtSize]; //FOLLOW集

stack<char> AnalyseStack; //分析栈

string AnalyseTable[VnSize][VtSize]; //预测分析表

string Production[ProductionSize]; //文法产生式

string inputBuffer; //输入缓冲区

int ip; //输入缓冲区指针

1. **函数与过程**

int mapVn(char c); //非终结符与数字对应

int mapVt(char c); //终结符与数字对应

void Initial(); //产生式、FIRST、FOLLOW初始化

void createAnalyseTable(); //创建预测分析表，算法4.2

string outStack(); //输出栈内容字符串

void error1(); //第一种错误，栈顶终结符与当前输入符不匹配

void error2(char X, char a); //第二种错误，表项空白

void error3(char X, char a); //第三种错误，表项为"synch"

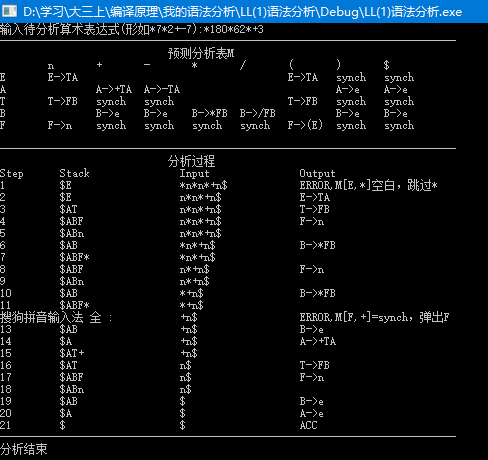
void LL1\_analyse(string w); //算法4.1,非递归预测分析方法

string inStringChange(string w); //输入数字转化为非终结符'n'

void TablePrint(); //输出预测分析表

1. 样例结果

**（1） 示例输入1，\*180\*62\*+3**



**结果分析**

通过示例1，能够看出错误处理能够识别出三种错误：

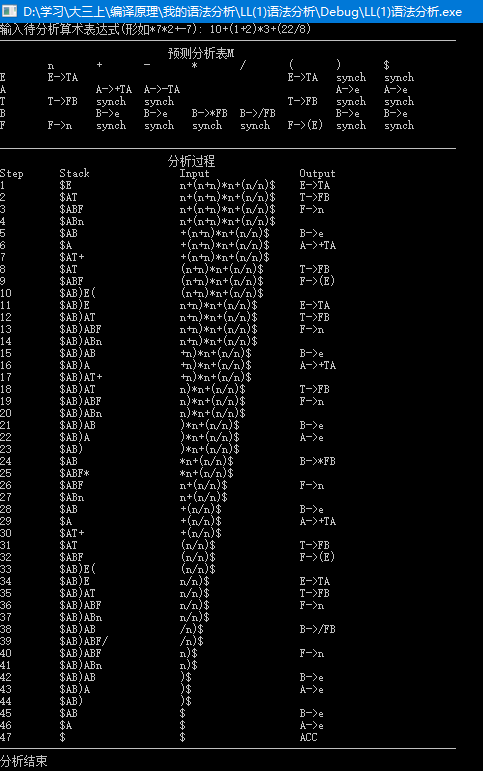
一种是X为终结符却不和当前输入符号a匹配，此时弹出栈顶终结符

第二种是预测分析表项为空，此时跳过该输入符号

第三种是预测分析表项为同步信息synch,此时弹出栈顶非终结符。

该程序能够很好的还原书本上的预测分析步骤，结果也是正确的。

**（2）** **示例输入2，10+(1+2)\*3+(22/8)**



1. 实验总结

这是编译原理与技术第二次程序设计实验，程序设计极大地参考并实现了课本上所给出的算法4.1与算法4.2，还有输入输出形式，采用了较为规范的编程形式，使程序具有良好的可读性，在不断地改进与debug后能够实现所给出的算术表达式的分析，这是令人欢欣鼓舞的！！

但是程序也有不足之处，程序中的文法改写，如消除左递归，提取左公因式，得出FIRST、FOLLOW集，都是通过人力运算得出，若能实现程序求出，将会变得更为智能。这是能够改进的。

这次实验获益匪浅，加深了对语法分析LL(1)文法的理解，也极大地激励了学习编译原理的热情，对课本内容的不断理解与探索，对debug的一丝不苟，都是对后面的学习将是极大的鼓励。

1. 源码附件

/\*\*

\*LL1\_Praser.cpp文件内容

\*作者:裴子祥

\*时间:2017.11.20

\*/

#include<iostream>

#include<vector>

#include<string>

#include<stack>

#include<iomanip>

using namespace std;

const int VnSize = 5; //非终结符个数

const int VtSize = 8; //终结符个数

const int ProductionSize = 10; //产生式个数

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

char Vt[VtSize] = { 'n','+','-','\*','/','(',')','$' };//终结符

vector<char> FIRST[VnSize]; //FIRST集

vector<char> FOLLOW[VtSize]; //FOLLOW集

stack<char> AnalyseStack; //分析栈

string AnalyseTable[VnSize][VtSize]; //预测分析表

string Production[ProductionSize]; //文法产生式

string inputBuffer; //输入缓冲区

int ip; //输入缓冲区指针

int mapVn(char c); //非终结符与数字对应

int mapVt(char c); //终结符与数字对应

void Initial(); //产生式、FIRST、FOLLOW初始化

void createAnalyseTable(); //创建预测分析表，算法4.2

string outStack(); //输出栈内容字符串

void error1(); //第一种错误，栈顶终结符与当前输入符不匹配

void error2(char X, char a); //第二种错误，表项空白

void error3(char X, char a); //第三种错误，表项为"synch"

void LL1\_analyse(string w); //算法4.1,非递归预测分析方法

string inStringChange(string w); //输入数字转化为非终结符'n'

void TablePrint(); //输出预测分析表

int main()//主函数

{

string s;

cout << "输入待分析算术表达式(形如\*7\*2+-7):";

cin >> s;

string w = inStringChange(s);

Initial();

createAnalyseTable();

TablePrint();

LL1\_analyse(w);

system("pause");

return 0;

}

int mapVn(char c)//非终结符与数字对应

{

switch (c)

{

case 'E':

return 0;

break;

case 'A':

return 1;

break;

case 'T':

return 2;

break;

case 'B':

return 3;

break;

case 'F':

return 4;

break;

default:

return -1;

break;

}

}

int mapVt(char c)//终结符与数字对应

{

switch (c)

{

case 'n':

return 0;

break;

case '+':

return 1;

break;

case '-':

return 2;

break;

case '\*':

return 3;

break;

case '/':

return 4;

break;

case '(':

return 5;

break;

case ')':

return 6;

break;

case '$':

return 7;

break;

default:

return -1;

break;

}

}

void Initial()

{

//消除左递归、提取左公因式后的产生式

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

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

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

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

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

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

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

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

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

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

//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('/');

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

{

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

{

AnalyseTable[i][j] = "";//预测分析表项置空

}

}

}

void createAnalyseTable()//创建预测分析表，算法4.2

{

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

{

int row = -1, col = -1, row2 = -1;//分析表横纵坐标

char RightHead = Production[i][3];

row = mapVn(Production[i][0]);

if (RightHead == 'n' || RightHead == '+' || RightHead == '-'

|| RightHead == '\*' || RightHead == '/' || RightHead == '('

|| RightHead == ')' || RightHead == '$')//为终结符时

{

col = mapVt(RightHead);

AnalyseTable[row][col] = Production[i];

}

else if (RightHead == 'E' || RightHead == 'A' || RightHead == 'T'

|| RightHead == 'B' || RightHead == 'F')//非终结符

{

int row2 = mapVn(RightHead);//非终结符

for (int j = 0; j < FIRST[row2].size(); ++j)

{

if (FIRST[row2][j] != 'e')//不是A->ε

{

col = mapVt(FIRST[row2][j]);

AnalyseTable[row][col] = Production[i];

}

else

{

for (int k = 0; k < FOLLOW[row].size(); ++k)

{

col = mapVt(FOLLOW[row][k]);

AnalyseTable[row][col] = Production[i];

}

}

}

}

else//A->ε

{

for (int k = 0; k < FOLLOW[row].size(); ++k)

{

col = mapVt(FOLLOW[row][k]);

AnalyseTable[row][col] = Production[i];

}

}

}

for (int i = 0; i < VnSize; ++i)//对非终结符遍历

{

int col = -1;

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

{

col = mapVt(FOLLOW[i][j]);//b∈FOLLOW(A)

if (AnalyseTable[i][col].empty())

{//若M[A,b]为空，填入"synch"

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

}

}

}

}

string outStack()//输出站内容

{

string s = "";

char ctemp;

stack<char> stemp;

while (!AnalyseStack.empty())

{

ctemp = AnalyseStack.top();

AnalyseStack.pop();

stemp.push(ctemp);

}

while (!stemp.empty())

{

ctemp = stemp.top();

s += ctemp;

AnalyseStack.push(ctemp);

stemp.pop();

}

return s;

}

void error1()//第一种错误，栈顶终结符与当前输入符不匹配

{

AnalyseStack.pop();//弹出栈顶

}

void error2(char X, char a)//第二种错误，表项空白

{

ip++;//空白，错误，跳过a

cout << "ERROR,M[" << X << ',' << a << "]空白，跳过" << a << endl;

}

void error3(char X, char a)//第三种错误，表项为"synch"

{

AnalyseStack.pop();//弹出栈顶

cout << "ERROR,M[" << X << ',' << a << "]=synch，弹出" << X << endl;

}

void LL1\_analyse(string w)//算法4.1,非递归预测分析方法

{

char X;//指向栈顶文法符号

char a;//ip所指向的输入符号

ip = 0;//w$指针，初始指向第一个符号

int step = 0;//记录步骤

inputBuffer = w;

AnalyseStack.push('$');

AnalyseStack.push('E');

cout << " 分析过程" << endl;

cout << left << setw(10) << "Step" << left << setw(20) << "Stack" << left << setw(20) << "Input" << left << setw(20) << "Output" << endl;

do

{

X = AnalyseStack.top();

a = inputBuffer[ip];

++step;

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

if (X == 'n' || X == '+' || X == '-' || X == '\*'

|| X == '/' || X == '(' || X == ')' || X == '$')//终结符或$

{

if (X == a)

{

AnalyseStack.pop();

ip++;

if (X == '$')

{

cout << "ACC";

cout << endl;

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

cout << "分析结束" << endl;

}

cout << endl;

}

else

{

error1();//第一种错误

}

}

else//非终结符

{

int rowVn = mapVn(X);

int colVt = mapVt(a);

if (rowVn == -1 || colVt == -1)

{

cout << "输入含有错误文法符号" << endl;

exit(0);

}

if (AnalyseTable[rowVn][colVt].empty())

{

error2(X, a);//第二种错误,空白

}

else if (AnalyseTable[rowVn][colVt] == "synch")

{

error3(X, a);//第三种错误synch

}

else//正常情况

{

if (AnalyseTable[rowVn][colVt][3] == 'e')

{

AnalyseStack.pop();

cout << AnalyseTable[rowVn][colVt] << endl;

}

else

{

AnalyseStack.pop();

for (int i = AnalyseTable[rowVn][colVt].length() - 1; i >= 3; --i)

{

AnalyseStack.push(AnalyseTable[rowVn][colVt][i]);

}

cout << AnalyseTable[rowVn][colVt] << endl;

}

}

}

} while (X != '$');

}

string inStringChange(string w)

{

string temp = "";

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

{

//cout << w[i] << endl;

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

{//是数字

++i;

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

{

++i;

}

temp += 'n';

--i;

}

else

{

temp += w[i];

}

}

temp += '$';

return temp;

}

void TablePrint()//输出预测分析表

{

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

cout << " 预测分析表M" << endl;

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

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

}

cout << endl;

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

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

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

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

}

cout << endl;

}

cout << endl;

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

}