**LL(1)语法分析程序报告**

**班级：2014211302**

**学号：2014211168**

**姓名：周尧棋**

**一、实验要求**

编写语法分析程序，实现对算术表达式的语法分析。要求所分析的算术表达式由如下文法产生。

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

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

F->(E)|num

在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。

**二、程序设计说明**

**1.整体说明**

LL(1)预测分析程序需要先构造预测分析表M，然后再利用一个分析栈和分析表对输入符号串进行自顶向下的分析。输入的文法表达式必须是消除了左递归之后的文法表达式。为得到预测分析表，首先要构造FIRST和FOLLOW集，根据书上算法4.2便可构造出预测分析表。最后利用算法4.1根据预测分析表和分析栈进行预测分析。

**2.获取产生式**

从键盘中输入产生式，存入到容器P中，对输入的产生式进行分析后得到产生式的起始符号、终结符、非终结符。输入的文法表达式必须消除左递归，并且在我的程序中用符号~代替了符号ε。

**3.FIRST集的和FOLLOW集的构造**

（1）数据结构说明

vector<char> VN; //非终结符号集合

vector<string> VT; //终结符号集合

char S; //起始符号S

string FIRST[20]; //每个非终结符号的FIRST集，各个符号之间用空格分隔开

string FOLLOW[20]; //每个非终结符号的FOLLOW集 ，各个符号之间用空格分隔开

int num = 0; //总共产生式的个数

struct Pro //结构体，分别存储产生式左边和右边

{

char left;

string right;

}pro[100];

（2）主要函数说明

string charToString(char c) //将char型转化为string

int findVN(vector<char> F, char c) //在VN集中寻找非终结符，返回下标

int findVT(vector<string> F, string c) //在VT集中寻找终结符，返回下标

void addToVT(string vt) //加入VT集，主要要处理num这种终结符

void split(string temp, vector<string> &str) //将以字符串表示的FIRST或者FOLLOW集按空格分隔开，存到vector中

string get\_FIRST(int index) //获取FIRST集，在该函数中会递归调用自身，所以输入参数为非终结符在VN容器中的下标，依照书上构造FIRST集的方法进行构造，稍微难处理的是出现num时的情况，需要往后继续读取字符再判断是否为终结符

void get\_follow(vector<string> &record) //对于每一个产生式，先找到能直接得出FOLLOW集的非终结符，即A->aBb情况，将FIRST(b)中的非ε元素加入FOLLOW(B)中。如果出现A->aB或者A->aBb但是ε属于FIRST(b)的情况，则存储记录AB于容器中，意思为在结束该函数后，在接下来的操作中，将FOLLOW(A)中的所有元素加入FOLLOW(B)中。

void get\_FOLLOW() //首先在文法起始符号的FOLLOW集中加入$，再通过容器中的记录，添加FOLLOW集中的元素

**4.预测分析表的构造**

预测分析表利用string型二维数组M[][]存储，函数void predict\_table(string M[][10])利用书上算法4.2进行构造。需要注意的是，对于产生式A->α，求FIRST(α)时看α的第一字符，如果α为终结符，则FIRST集就是本身；如果α是非终结符，且FIRST(α)中还包含ε，则还要继续往后读一个符号再求后面符号的FIRST集；

**5.LL(1)预测分析程序**

函数void predict\_anylsis(string M[][10])利用预测分析表进行自顶向下的预测分析，并对分析栈、输入串和输出进行打印。

**三、源程序**

#include<iostream>

#include<cstdlib>

#include<vector>

#include<sstream>

#include<string>

#include<stack>

using namespace std;

vector<char> VN; //非终结符号集合

vector<string> VT; //终结符号集合

vector<string> P; //产生式集合

char S; //起始符号S

string FIRST[20]; //每个非终结符号的FIRST集

string FOLLOW[20]; //每个非终结符号的FOLLOW集

int num = 0; //总共产生式的个数

struct Pro //结构体，存储产生式

{

char left;

string right;

}pro[100];

string charToString(char c)

//将char型转换为string

{

string str;

stringstream stream;

stream << c;

str = stream.str();

return str;

}

int findVN(vector<char> F, char c)

//在VN集中寻找非终结符，返回下标

{

vector<char>::iterator it;

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

{

if ((\*it) == c)

{

return it - F.begin();

}

}

return -1;

}

int findVT(vector<string> F, string c)

//在VT集中寻找终结符，返回下标

{

vector<string>::iterator it;

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

{

if ((\*it) == c)

{

return it - F.begin();

}

}

return -1;

}

void addToVT(string vt)

//加入VT集

{

if (vt.length() > 0)

{

if (vt == "num")

{

if (findVT(VT,vt) < 0)

{

VT.push\_back(vt);

}

}

else

{

for (int j = 0; j < vt.length(); j ++)

{

if (findVT(VT,charToString(vt[j])) < 0)

{

VT.push\_back(charToString(vt[j]));

}

}

}

}

}

void getP()

//输入产生式

{

string s;

string temp;

string vt;

cout << "用~代替了ε！" << endl;

cout << "输入产生式：（结束输入over）" << endl;

cin >> s;

while (s != "over")

{

temp.clear();

vt.clear();

if(s.length() > 1)

{

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

{

if(s[i] >= 'A' && s[i] <= 'Z')

{

if (findVN(VN,s[i]) < 0)

{

VN.push\_back(s[i]);

}

if (i != 0)

{

temp += s[i];

addToVT(vt);

vt.clear();

//cout << temp << endl;

}

}

else if ((s[i] == '-' && s[i+1] == '>') || (s[i] == '>' && s[i-1] == '-'))

{

continue;

}

else if (s[i] == '|')

{

addToVT(vt);

vt.clear();

string j = "->";

P.push\_back(s[0] + j + temp);

temp.clear();

}

else

{

vt += s[i];

temp += s[i];

}

if (i == s.length() - 1)

{

addToVT(vt);

vt.clear();

string j = "->";

P.push\_back(s[0] + j + temp);

temp.clear();

}

}

}

cout << "输入产生式：（结束输入over）" << endl;

cin >> s;

}

vector<char>::iterator it;

vector<string>::iterator it1;

cout << "VN:" ;

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

{

cout << \*it << " ";

}

cout << endl;

cout << "VT:" ;

for (it1 = VT.begin(); it1 != VT.end(); it1 ++)

{

cout << \*it1 << " ";

}

cout << endl;

}

void split(string temp, vector<string> &str)

//将以字符串表示的FIRST或者FOLLOW集按空格分隔开，存到vector中

{

str.clear();

int pos = 0;

while(1)

{

string temp\_s = "";

pos = temp.find(" ");

if(pos == temp.npos)

{

temp\_s = temp.substr(0,temp.length());

str.push\_back(temp\_s);

break;

}

temp\_s = temp.substr(0,pos);

temp.erase(0,pos+1);

str.push\_back(temp\_s);

}

str.pop\_back();

}

string get\_FIRST(int index)

//获取FIRST集

{

char c = VN[index];

int m = 0;

string right\_s;

vector<string> str;

vector<string>::iterator it;

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

{

if (pro[i].left == c)

{

if (pro[i].right[0] >= 'A' && pro[i].right[0] <= 'Z')

//表达式右边第一个为非终结符号，X->Y...情况

{

if (findVN(VN,pro[i].right[0]) >= 0)

//在非终结符表中

{

string temp = get\_FIRST(findVN(VN,pro[i].right[0])); //temp为右边第一个非终结符的first集

//cout << temp << endl;

str.clear();

split(temp,str);

//将first(Y)中的非~元素加入first(X)中

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

{

if (\*it != "~" && FIRST[index].find(\*it) == FIRST[index].npos)

{

FIRST[index] += \*it;

FIRST[index] += ' ';

}

}

int count = 1;

if (count < pro[i].right.length())

{

while (temp.find('~') != temp.npos && count < pro[i].right.length())

//如果first(Y)中有~元素，

{

//cout << pro[i].right[count] << endl;

if (findVT(VT,charToString(pro[i].right[count])) >= 0)

{

if (FIRST[index].find(pro[i].right[count]) == FIRST[index].npos)

{

FIRST[index] += pro[i].right[count];

FIRST[index] += ' ';

break;

}

}

else

{

temp = get\_FIRST(findVN(VN,pro[i].right[count]));

count ++;

}

str.clear();

split(temp,str);

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

{

if (\*it != "~" && FIRST[index].find(\*it) == FIRST[index].npos)

{

FIRST[index] += \*it;

FIRST[index] += ' ';

}

}

}

}

}

}

else

//表达式右边第一个为终结符号

{

if (findVT(VT,charToString(pro[i].right[0])) >= 0)

//X->a...，其中a属于VT

{

if (FIRST[index].find(pro[i].right[0]) == FIRST[index].npos)

{

FIRST[index] += pro[i].right[0];

FIRST[index] += ' ';

}

}

else

//处理为终结符号为num的情况

{

while (right\_s != "num")

{

right\_s += pro[i].right[m];

m++;

}

if (FIRST[index].find("num") == FIRST[index].npos)

{

FIRST[index] += right\_s;

FIRST[index] += ' ';

}

right\_s.clear();

}

}

}

}

return FIRST[index];

}

void get\_follow(vector<string> &record)

{

int i, j, m;

vector<string> str;

for (i = 0; i < num; i ++)

{

for (j = 0; j < pro[i].right.length(); j ++)

{

//找到一个非终结符号B

int n = findVN(VN,pro[i].right[j]); //VN的位置

if (n >= 0)

{

if (j < pro[i].right.length() - 1)

//B不为最后一个符号

{

if (findVT(VT,charToString(pro[i].right[j+1])) >= 0)

//B后面接的是终结符，A->aBb，加入follow(B)中

{

if (FOLLOW[n].find(pro[i].right[j+1]) == FOLLOW[n].npos)

{

FOLLOW[n] += pro[i].right[j+1];

FOLLOW[n] += ' ';

}

}

//B后面接的是非终结符，A->aBC，将first(C)的非~符号加入follow(B)中

else if (findVN(VN,pro[i].right[j+1]) >= 0)

{

string temp = FIRST[findVN(VN,pro[i].right[j+1])];

int flag = 0;

vector<string>::iterator it;

str.clear();

split(temp,str);

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

{

if (\*it == "~")

//若first(C)中有~元素，把follow(A)所有元素加入follow(B)中

{

flag = 1;

}

if (\*it != "~" && FOLLOW[n].find(\*it) == FOLLOW[n].npos)

{

FOLLOW[n] += \*it;

FOLLOW[n] += ' ';

}

}

if (flag == 1)

{

string r ;

r += pro[i].left;

r += pro[i].right[j];

record.push\_back(r);

}

}

}

else

//产生式为A->aB的情况，直接将follow(A)中的元素加入follow(B)中

{

string r ;

r += pro[i].left;

r += pro[i].right[j];

record.push\_back(r);

}

}

}

}

}

void get\_FOLLOW()

//获取FOLLOW集

{

cout << "请输入该文法的起始符号：" << endl;

cin >> S;

FOLLOW[findVN(VN,S)] += "$ ";

vector<string> record;

vector<string> str;

get\_follow(record);

vector<string>::iterator it;

vector<string>::iterator it1;

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

{

string s = (\*it);

char A = s[0];

char B = s[1];

string temp = FOLLOW[findVN(VN,A)];

str.clear();

split(temp,str);

for (it1 = str.begin(); it1 != str.end(); it1 ++)

{

if (FOLLOW[findVN(VN,B)].find(\*it1) == FOLLOW[findVN(VN,B)].npos)

{

FOLLOW[findVN(VN,B)] += \*it1;

FOLLOW[findVN(VN,B)] += ' ';

}

}

}

}

void predict\_table(string M[][10])

//构造预测分析表，存入M[][]中

{

string s;

s += '$';

VT.push\_back(s);

vector<string> First;

vector<string> Follow;

vector<string>::iterator it;

bool flag = false;

int i,j,a,b;

for (i = 0; i < num; i ++)

{

flag = false;

a = findVN(VN,pro[i].left); //表达式左边非终结符在VN中的位置

for (j = 0; j < pro[i].right.length(); j ++)

{

int vn = findVN(VN,pro[i].right[j]);

int vt = findVT(VT,pro[i].right.substr(0,j+1));

if (vn >= 0)

{

split(FIRST[vn],First);

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

{

if (\*it == "~")

{

flag = true;

}

b = findVT(VT,\*it);

M[a][b] += pro[i].left;

M[a][b] += "->" + pro[i].right ;

}

if (!flag)

{

break;

}

}

else if (vn < 0 && vt >= 0)

{

M[a][vt] += pro[i].left;

M[a][vt] += "->" + pro[i].right ;

if (VT[vt] == "~")

{

flag = true;

}

break;

}

}

if (flag)

{

split(FOLLOW[a],Follow);

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

{

b = findVT(VT,\*it);

M[a][b] += pro[i].left;

M[a][b] += "->" + pro[i].right ;

}

}

}

for (i = 0; i < VN.size(); i ++)

{

split(FOLLOW[i],Follow);

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

{

int j = findVT(VT,\*it);

if (M[i][j].empty())

{

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

}

}

}

cout << "预测分析表为：" << endl;

cout << '\t' ;

for (j = 0; j < VT.size(); j ++)

{

if (VT[j] != "~")

{

cout << VT[j] << '\t';

}

}

cout << endl;

for (i = 0; i < VN.size(); i ++)

{

cout << VN[i] << '\t';

for (j = 0; j < VT.size(); j ++)

{

if (VT[j] != "~")

{

cout << M[i][j] << '\t';

}

}

cout << endl;

}

}

void predict\_anylsis(string M[][10])

{

string input;

int i = 0;

int vt,vn;

cout << "输入符号串：" << endl;

cin >> input;

cout << endl;

input += "$";

stack<string> St;

St.push("$");

string in\_stack;

in\_stack += S;

St.push(in\_stack);

cout << "分析过程为：" << endl;

cout << "栈：" << S << " $" << '\t' << "输入：" << input << '\t' << "输出：" << endl;

while(St.top() != "$")

{

vt = findVT(VT,St.top());

vn = findVN(VN,St.top()[0]);

string carry;

stack<string> R;

if (vt >= 0)

{

//cout << St.top() << endl;

if (St.top() == input.substr(i,St.top().length()))

{

i += St.top().length();

St.pop();

}

else //栈顶符号为终结符号，与当前输入符号不匹配

{

cout << "error! 栈顶符号与输入符号不匹配，从栈中弹出终结符" << endl;

St.pop();

}

//栈的输出

cout << "栈：";

while(!St.empty())

{

carry = St.top();

St.pop();

R.push(carry);

cout << carry << " ";

}

while(!R.empty())

{

carry = R.top();

R.pop();

St.push(carry);

}

//输入串

cout << '\t' << "输入：";

cout << input.substr(i);

//输出

cout << '\t' << "输出：" << endl;

//system("pause");

}

else if (vt < 0 && vn >= 0)

{

string tmp;

tmp += input[i];

int j = findVT(VT,tmp);

int k = i;

while (j < 0)

{

tmp += input[++k];

j = findVT(VT,tmp);

}

//cout << "tmp: " << tmp << endl;

tmp.clear();

string right;

if (!M[vn][j].empty())

{

if (M[vn][j] != "synch")

{

right = M[vn][j].substr(3);

}

else

{

right = "synch";

}

}

else

{

right.clear();

}

if (!right.empty() && right != "synch")

{

St.pop();

for (int m = 0; m < right.length(); m ++)

{

tmp += right[m];

int n1 = findVT(VT,tmp);

int n2 = findVN(VN,tmp[0]);

if (n1 >= 0 || n2 >= 0)

{

if (tmp != "~")

{

R.push(tmp);

}

tmp.clear();

}

}

while(!R.empty())

{

St.push(R.top());

R.pop();

}

}

else if (right.empty())

{

i += VT[j].length();

cout << "error! M[][]为空，向前移动指针" << endl;

}

else

{

St.pop();

cout << "error! M[][]为synch，从栈中弹出非终结符" << endl;

}

//栈的输出

cout << "栈：";

while(!St.empty())

{

carry = St.top();

St.pop();

R.push(carry);

cout << carry << " ";

}

while(!R.empty())

{

carry = R.top();

R.pop();

St.push(carry);

}

//输入串

cout << '\t' << "输入：";

cout << input.substr(i);

//输出

cout << '\t' << "输出：";

cout << M[vn][j] << endl;

//system("pause");

}

}

}

int main()

{

getP();

string M[10][10];

vector<string>::iterator it;

int i;

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

{

pro[num].left = (\*it)[0];

pro[num].right = (\*it).substr(3);

num ++;

}

cout << "FIRST集：" << endl;

for(i = 0; i < VN.size(); i++)

{

cout << VN[i] << ": " << get\_FIRST(i) << endl;

}

cout << endl;

get\_FOLLOW();

cout << "FOLLOW集：" << endl;

for(i = 0; i < VN.size(); i++)

{

cout << VN[i] << ": " << FOLLOW[i] << endl;

}

cout << endl;

predict\_table(M);

cout << endl;

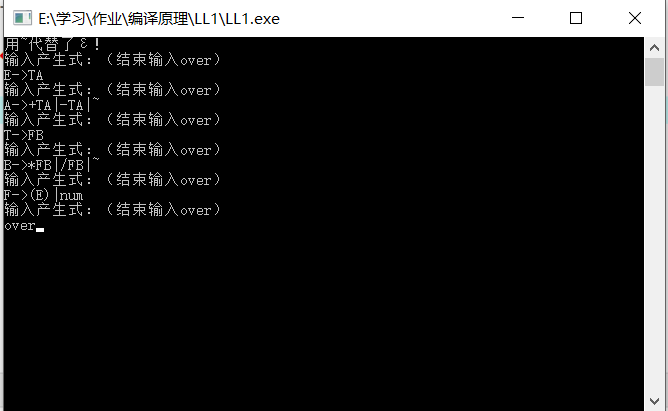
predict\_anylsis(M);

system("pause");

return 0;

}

**四、程序测试说明**

****

首先输入消除了左递归的文法（键盘输入时ε用~代替）

E->TA

A->+TA|-TA|ε

T->FB

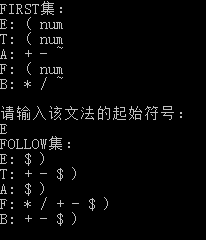
B->\*FB|/FB|ε

F->(E)|num

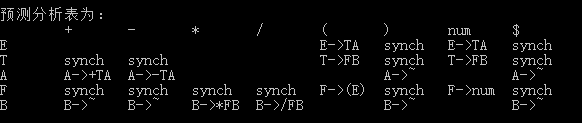
getP()函数根据输入的产生式得到VN和VT集



get\_FIRST()和get\_FOLLOW函数得到FIRST集和FOLLOW集

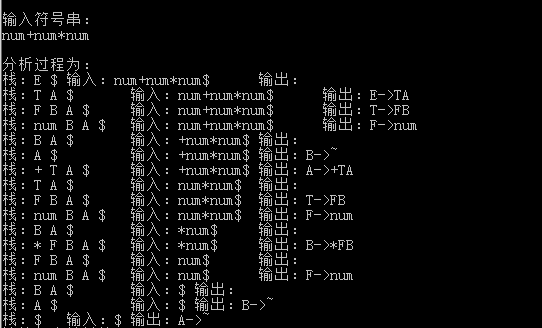


predict\_table()函数得到预测分析表

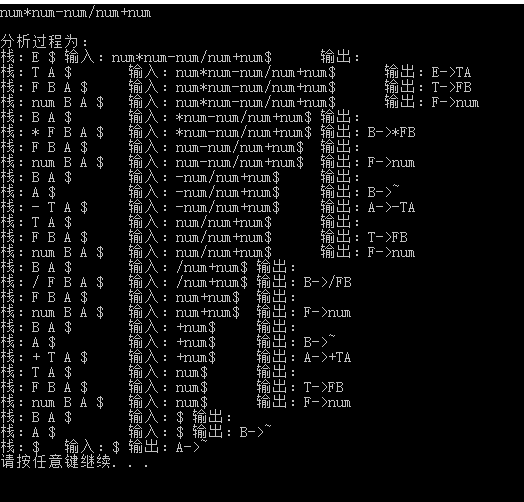


最后根据predict\_anylsis()函数得到分析过程

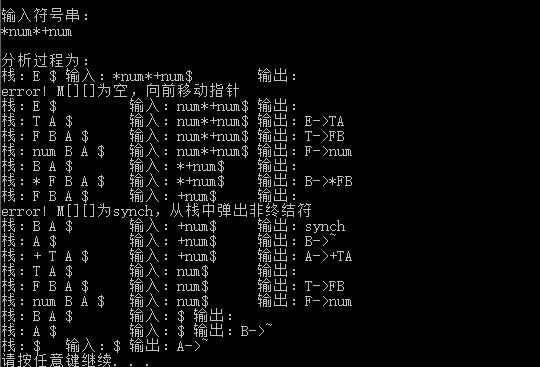
当输入符号串为num+num\*num时，没有错误提示且最后栈中只剩$，所以输入符号串能由输入的文法产生。



当输入符号串为num\*num-num/num+num时，没有错误提示且最后栈中只剩$，所以输入符号串能由输入的文法产生。



当输入符号串为\*num\*+num时，分析时出提示错误，并跳过一些输入的字符继续进行分析，如果栈顶符号为终结符号，但它与当前输入符号不匹配，则将此终结符从栈顶弹出；如果栈顶符号为非终结符号A，当前输入符号为a，若M[A][a]为空白则向前移动指针，若M[A][a]为synch则从栈顶弹出A。



**五、心得与体会**

此次LL(1)语法分析程序的编写历经磨难，但我在其中收获颇丰。首先是在求FIRST集合FOLLOW集合时，经过反复的思考和尝试后才得到了适当的数据结构来存储表达式和两个集合，又为了输出美观，不停地对存储的结构进行操作也让我受益匪浅。我也加强了对求FIRST集和FOLLOW集的理解，能更加熟练地手动求出。在求预测分析表时，我发现对于产生式A->α，求FIRST(α)时看α的第一字符，如果α为终结符，则FIRST集就是本身；如果α是非终结符，且FIRST(α)中还包含ε，则还要继续往后读一个符号再求后面符号的FIRST集。第二种情况在平时的作业练习中没有遇见过，我也在编码时曾忽略了，这是要值得注意的地方。加入了synch信息后的预测分析表更为完善，加强了处理能力。