LL(1)语法分析程序

1. 题目及要求

编写LL(1)语法分析程序，实现对算术表达式的语法分析。要求所分析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

1. 编程实现预测分析表的构造
2. 编程实现LL(1)预测分析程序
3. 程序设计说明
4. 消除文法的左递归（这里直接在程序中给出，未通过程序计算得到）

新的文法如下：

E -> TW

W -> +TW

W -> -TW

W -> e

T -> FV

V -> \*FV

V -> /FV

V -> e

F -> (E)

F -> num（后面程序输出结果F -> n代表此产生式）

2．构造预测分析表

(1) 构造文法的FIRST和FOLLOW集

|  |  |  |
| --- | --- | --- |
|  | FIRST | FOLLOW |
| E | (, num | $, ) |
| W | +, -, e | $, ) |
| T | (,, num | +, -, $, ) |
| V | \*, /, e | +, -, $, ) |
| F | (, num | +, -, \*, /, ),$ |

(2) 预测分析表构造程序

for (文法G的每一个产生式A -> α ){

for ( 每个终结符号a∈ FIRST(α) ) 把A ->α放入表项M[A, a]中

if ( e ∈ FIRST(α) )

for ( 每个b ∈ FOLLOW(A) ) 把A ->α放入表项M[A, b]中

}

for ( 所有无定义的表项M[A, a] ) 标上错误标志

3. 预测分析程序

do {

令X是栈顶文法符号，a是ip所指向的输入符号

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

if ( M[A, a] == X -> Y1Y2...Yk ){

从栈顶弹出X;

依次把Yk, Yk-1, ...,Y2, Y1压入栈;

输出产生式X -> Y1Y2...Yk;

}

else error();

}while ( X != $ )

1. 源代码

#include<iostream>

#include<string>

#include<vector>

using namespace std;

string orig\_G[8] = {

"E->E+T", "E->E-T", "E->T", "T->T\*F", "T->T/F", "T->F", "F->(E)", "F->n"};

string no\_left\_reg\_G[10] = {

"E->TW", "W->+TW", "W->-TW", "W->e", "T->FV", "V->\*FV", "V->/FV", "V->e", "F->(E)", "F->n"};

string FIRST[5] = {

"E:(,n", "W:+,-,e", "T:(,n", "V:\*,/,e", "F:(,n"};

string FOLLOW[5] = {

"E:$,)", "W:$,)", "T:+,-,$,)", "V:+,-,$,)", "F:\*,/,+,-,$"};

char VN[5] = {'E', 'W', 'T', 'V', 'F'};

char VT[8] = {'+', '-', '\*', '/', '(', ')', 'n', '$'};

typedef struct entry\_M{

int occupied;

char VN;

char VT;

string production;

}entry\_M;

entry\_M table[40];

bool isVN( char ch )

{

for ( int i = 0; i < 5; i++ ){

if ( VN[i] == ch ) return true;

}

return false;

}

bool isVT( char ch )

{

for ( int i = 0; i < 8; i++ ){

if ( VT[i] == ch ) return true;

}

return false;

}

int get\_VN\_loc( char ch )

{

for ( int i = 0; i < 5; i++ ){

if ( VN[i] == ch ) return i;

}

return -1;

}

int get\_VT\_loc( char ch )

{

for ( int i = 0; i < 8; i++ ){

if ( VT[i] == ch ) return i;

}

return -1;

}

void build\_table()

{

for ( int i = 0; i < 40; i++ ){

table[i].occupied = 0;

}

for ( int i = 0; i < 10; i++ ){

if ( isVN(no\_left\_reg\_G[i].at(3)) ){

char VN = no\_left\_reg\_G[i].at(3);

int row = get\_VN\_loc(no\_left\_reg\_G[i].at(0));

for ( int j = 0; j < 5; j++ ){

if ( FIRST[j].at(0) == VN ){

for ( int k = 2; k < FIRST[j].length(); k++ ){

if ( FIRST[j][k] != ',' && FIRST[j][k] != 'e' ){

char VT = FIRST[j][k];

int col = get\_VT\_loc(VT);

table[row\*8 + col].VN = no\_left\_reg\_G[i].at(0);

table[row\*8 + col].VT = VT;

table[row\*8 + col].production = no\_left\_reg\_G[i];

table[row\*8 + col].occupied = 1;

}

else if ( FIRST[j][k] == 'e' ){

for ( int s = 0; s < 5; s++ ){

if ( FOLLOW[s].at(0) == no\_left\_reg\_G[i].at(0) ){

for ( int t = 2; t < FOLLOW[s].length(); t++ ){

if ( FOLLOW[s][t] != ',' ){

char VT = FOLLOW[s][t];

int col = get\_VT\_loc(VT);

table[row\*8 + col].VN = no\_left\_reg\_G[i].at(0);

table[row\*8 + col].VT = VT;

table[row\*8 + col].production = no\_left\_reg\_G[i];

table[row\*8 + col].occupied = 1;

}

}

}

}

}

}

}

}

}

else if ( isVT(no\_left\_reg\_G[i].at(3)) ){

char VT = no\_left\_reg\_G[i].at(3);

int col = get\_VT\_loc(VT);

int row = get\_VN\_loc(no\_left\_reg\_G[i].at(0));

table[row\*8 + col].VN = no\_left\_reg\_G[i].at(0);

table[row\*8 + col].VT = VT;

table[row\*8 + col].production = no\_left\_reg\_G[i];

table[row\*8 + col].occupied = 1;

}

else if ( no\_left\_reg\_G[i].at(3) == 'e' ){

char VT = no\_left\_reg\_G[i].at(3);

int row = get\_VN\_loc(no\_left\_reg\_G[i].at(0));

for ( int s = 0; s < 5; s++ ){

if ( FOLLOW[s].at(0) == no\_left\_reg\_G[i].at(0) ){

for ( int t = 2; t < FOLLOW[s].length(); t++ ){

if ( FOLLOW[s][t] != ',' ){

char VT = FOLLOW[s][t];

int col = get\_VT\_loc(VT);

table[row\*8 + col].VN = no\_left\_reg\_G[i].at(0);

table[row\*8 + col].VT = VT;

table[row\*8 + col].production = no\_left\_reg\_G[i];

table[row\*8 + col].occupied = 1;

}

}

}

}

}

}

}

void error()

{

cout << "Unaccepted senntence!" << endl;

}

int main()

{

build\_table();

cout << "The analyse table is:" << endl;

for ( int i = 0; i < 6; i++ ){

for ( int j = 0; j < 9; j++ ){

if ( i == 0 && j == 0) cout << '\t';

else if ( i == 0 ){

cout << VT[j-1] << '\t';

}

else if ( j == 0 ){

cout << VN[i-1] << '\t';

}

else{

cout << table[(i-1)\*8+(j-1)].production << '\t';

}

}

cout << endl;

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

}

string buffer;

string buffera;

int ip = 0;

int top = 1;

char X;

char a;

vector<char> stack;

stack.push\_back('$');

stack.push\_back('E');

cout << "Please input the sentence:" << endl;

cin >> buffer;

buffera = buffer + "$";

cout << "Stack" << '\t' << "Input" << '\t' << "Output"<< endl;

do{

X = stack[top];

a = buffera.at(ip);

if ( isVT(X) ){

if ( X == a ){

stack.pop\_back();

top--;

ip++;

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

cout << stack[i];

cout << '\t';

for ( int i = ip; i < buffera.length(); i++ )

cout << buffera.at(i);

cout << '\t';

}

else{

error();

break;

}

}

else{

int row = get\_VN\_loc(X);

int col = get\_VT\_loc(a);

if ( row < 0 || col < 0 ){

error();

break;

}

else{

if ( table[row\*8 + col].occupied != 0 ){

stack.pop\_back();

top--;

string production = table[row\*8 + col].production;

int upper = production.length();

if ( production.at(upper-1) != 'e' ){

for ( int i = upper-1; i >=3; i-- ){

stack.push\_back(production.at(i));

top++;

}

}

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

cout << stack[i];

cout << '\t';

for ( int i = ip; i < buffera.length(); i++ )

cout << buffera.at(i);

cout << '\t';

cout << production;

}

else{

error();

break;

}

}

}

cout << endl;

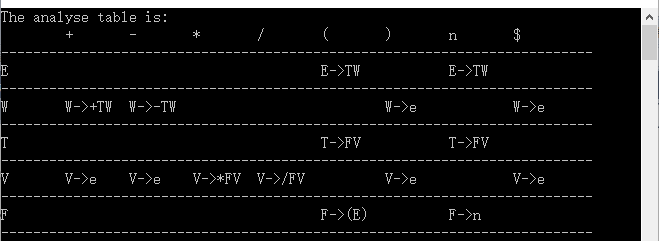
}while(X != '$');

system("pause");

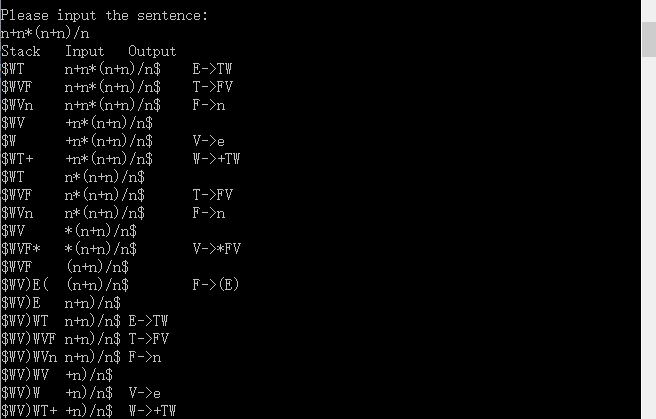
return 0;

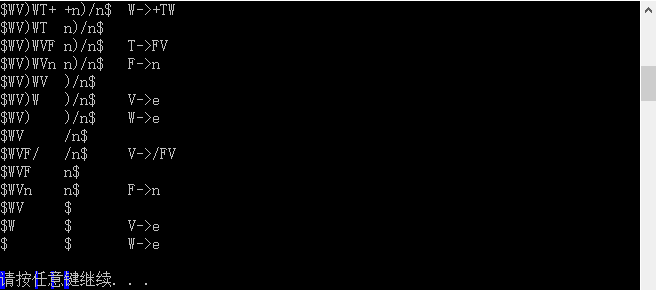
}

1. 测试及结果
2. 预测分析表的输出结果



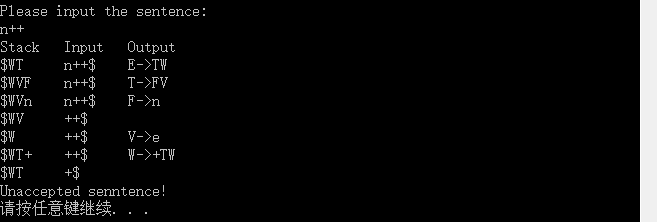
1. 预测分析程序输入及输出结果
2. 输入合法

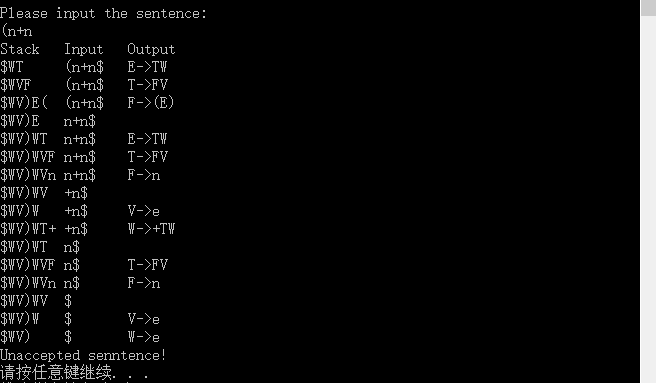




分别显示了栈情况、输入串和输出产生式

1. 输入非法





在分析错误时终止程序，输出不可接受的信息。