## 一 Mini C - BNF 范式

来源： <http://blog.csdn.net/pinbodexiaozhu/article/details/25394417>

**（一）文法表达式**

1. program  →  declaration-list

2. declaration-list → declaration-list  declaration | declaration

3. declaration → var-declaration | fun-declaration

4. var-declaration; → type-specifier ID  | type-specifier ID [ NUM ]  ;

5. type-specifier → int | void | float

6. fun-declaration → type-specifier ID(params)  compound-stmt

7. params → params-list | void

8. param-list → param-list,param | param

9. param → type-specifier ID

10. compound-stmt → { local-declarations statement-list }

11. local-declarations → local-declarations var-declaration | empty

12. statement-list → statement-liststatement | empty

13. statement → expression-stmt | compound-stmt | selection-stmt  | iteration-stmt | return-stmt

14. expression-stmt → expression;|;

15. selection-stmt → if (expression) statement  | if (expression) statement else statement

16. iteration-stmt → while (expression) statement

17. return-stmt → return; | return expression;

18. expression → var = expression | simple-expression

19. var → ID | ID[expression]

20. simple-expression → additive-expression relop additive-expression |  additive-expression

21. relop → <= | < | > | >= | = = | != | && | ||

22. additive-expression → additive-expression addop term | term  23. addop → + | -

24. term → term mulopfactor | factor  25. mulop → \* | /

26. factor → (expression) | var | call | NUM  27. call → ID (args)

28. args → arg-list |empty

29. arg-list → arg-list,expression | expression

**（二）文法的解释说明**

**1.program → declaration-list**

**2.declaration-list → declaration-list declaration | declaration**

**3.declaration → var-declaration | fun-declaration**

程序由声明的列表(或序列)组成，声明可以是函数或变量声明，顺序是任意的。至少必须有 一个声明。接下来是语义限制(这些在C中不会出现)。所有的变量和函数在使用前必须声明 (这避免了向后backpatching引用)。程序中最后的声明必须是一个函数声明，名字为main。

**4.var-declaration → type-specifier ID ; | type-specifier ID[NUM];**

**5.type-specifier → int | void | float | char**

变量声明或者声明了简单的整数或浮点类型变量，或者是基类型为整数或浮点的数组变量， 索引范围从0到NUM-1。注意，在C1中仅有的基本类型是整型和空类型。在一个变量声明中，只能使用类型指示符int。void用于函数声明(参见下面)。也要注意，每个声明只能声明一个变量。

**6.fun-declaration → type-specifier ID(params) compound-stmt**

**7.params → param-list | void**

**8.param-list → param-list , param | param**

**9.param → type-specifier ID**

函数声明由返回类型指示符(type-specifier)、标识符(ID)以及在圆括号内的用逗号分开的参数列表(params)组成，后面跟着一个复合语句(compound-stmt)，是函数的代码。

如果函数的返回类型是void，那么函数不返回任何值(即是一个过程)。函数的参数可以是void(即没有参数)，或者一列描述函数的参数。参数后面 跟着方括号是数组参数，其大小是可变的。简单的整型参数由值传递。数组参数由引用来传递(也就是指针)，在调用时必须通过数组变量来匹配。注意，类型“函数”没有参数。一个 函数参数的作用域等于函数声明的复合语句，函数的每次请求都有一个独立的参数集。函数 可以是递归的(对于使用声明允许的范围)。

**10.compound-stmt → { local-declarations statement-list }**

复合语句由用花括号围起来的一组声明(local-declarations)和语句(statement-list)组成。复合语句通过用给定的顺序执行语句序列来执行。局部声明的作用域等于复合语句的语句列表，并代替任何全局声明.

**11.local-declarations → local-declarations var-declaration | empty**

**12.statement-list → statement-list statement | empty**

注意声明和语句列表都可以是空的(非终结符empty表示空字符串，有时写作ε。)

**13.statement → expression-stmt  | compound-stmt  | selection-stmt  | iteration-stmt  | return-stmt**

**14.expression-stmt → expression; | ;**

表达式语句有一个可选的且后面跟着分号的表达式。这样的表达式通常求出它们一方的结 果。因此，这个语句用于赋值和函数调用。

**15.selection-stmt → if (expression) statement  | if (expression) statement else statement**

if语句有通常的语义：表达式进行计算；非0值引起第一条语句的执行；0值引起第二条语句的执行，如果它存在的话。这个规则导致了典型的悬挂else二义性，可以用一种标准的方法解决：else部分通常作为当前if的一个子结构立即分析(“最近嵌套”非二义性规则)。

**16.iteration-stmt → while (expression) statement**

while语句是C－中唯一的重复语句。它重复执行表达式，并且如果表达式的求值为非0，则 执行语句，当表达式的值为0时结束。

**17.return-stmt → return; | return expression** **;**

返回语句可以返回一个值也可无值返回。函数没有说明为void就必须返回一个值。函数声明为void就没有返回值。return引起控制返回调用者(如果它在main中，则程序结束)。

**18.expression → var=expression | simple-expression**

**19.var→ ID | ID[expression]**

表达式是一个变量引用，后面跟着赋值符号(等号)和一个表达式，或者就是一个简单的表达 式。赋值有通常的存储语义：找到由var表示的变量的地址，然后由赋值符右边的子表达式 进行求值，子表达式的值存储到给定的地址。这个值也作为整个表达式的值返回。

var是简单的(整型)变量或下标数组变量。负的下标将引起程序停止(与C不同)。然而，不进行下标 越界检查。

var表示C1比C的进一步限制。在C中赋值的目标必须是左值(l-value)，左值是可以由许多操作获得的地址。在C1中唯一的左值是由var语法给定的，因此这个种类按照句法进行检查， 代替像C中那样的类型检查。故在C1中指针运算是禁止的。

**20.simple-expression → additive-expression relop additive-expression  |**

**additive-expression**

**21.relop → <= | < | > | >= | == | != | && | ||**

简单表达式由无结合的关系操作符组成(即无括号的表达式仅有一个关系操作符)。简单表达 式在它不包含关系操作符时，其值是加法表达式的值，或者如果关系算式求值为ture，其值 为1，求值为false时值为0。

**22.additive-expression → additive-expression addop term | term**

**23.addop → + | -**

**24.term → term mulop factor | factor**

**25.mulop → \* | /**

加法表达式和项表示了算术操作符的结合性和优先级。/符号表示整数除；即任何余数都被 截去。

**26.factor → (expression) | var | call | NUM | DECIMAL**

因子是围在括号内的表达式；或一个变量，求出其变量的值；或者一个函数调用，求出函数 的返回值；或者一个NUM，其值由扫描器计算。数组变量必须是下标变量，除非表达式由单个ID组成，并且以数组为参数在函数调用中使用(如下所示)。

**27.call → ID(args)**

**28.args → arg-list | empty**

**29.arg-list → arg-list , expression | expression**

函数调用的组成是一个ID(函数名)，后面是用括号围起来的参数。参数或者为空，或者由逗 号分割的表达式列表组成，表示在一次调用期间分配的参数的值。函数在调用之前必须声明， 声明中参数的数目必须等于调用中参数的数目。函数声明中的数组参数必须和一个表达式匹 配，这个表达式由一个标识符组成表示一个数组变量。

本文法中，有以下终结符：

标识符 ID

整数数字 NUM

小数数字 DECIMAL

保留字 If else while return

操作符：+ - \* / < > <= >= == != || &&

分隔符 , ; () [] {}

## 构建LL(1)文法

考虑到LR（1）文法项目集太多，不适合手工操作，所以选择了LL（1）文法.

为便于书写，用每个单词的首字母代表整个含义

第一步，提取左公因子，消除左递归

**（一）过程如下：**

1. PR → DL

2. **DL → DL D | D**

**=>消除左递归 DL →D DL |** epsilon

3. D → VD | FD

**4. VD → TS ID | TS ID [NUM]**

**=>提取左公因子 VD → TS ID VD‘ ;**

**VD‘ → epsilon**

**VD‘ → [NUM]**

6. FD → TS ID (Ps) CS

**=>3,4,6一起，合并，并提取左公因子 D → TS ID D‘**

**D‘ → epsilon | [NUM] | (P) CS**

5. TS → int | void | float

7. Ps → PL | **epsilon**

**8. PL → PL , P | P**

**=>消除左递归 PL→P PL‘**

**P-L‘ →, P P-L‘ | epsilon**

9. P → T-S ID

10. C-S → { local-D S-L }

**11. local-D → local-D V-D | empty**

**=>消除左递归 local-D → V-D local-D | empty**

**12. S-L → S-L S | empty**

**=>消除左递归 S-L → S S-L | empty**

13. S → E-S | C-S | selection-S | iteration-S | return-S

14. E-S → E;|;

**15. selection-S → if (E) S | if (E) S else S**

**=>提取左公因子 selection-S → if (E) S selection-S’**

**selection-S’ → else S | epsilon**

16. iteration-S → while (E) S

**17. return-S → return; | return E;**

**=>提取左公因子 return-S → return return-S’**

**return-S’ → ; | E;**

18. E → V = E | simple-E

**19. V → ID | ID[E]**

**=>提取左公因子 V → ID V’**

**V’-> [E] | epsilon**

**20. simple-E → additive-E relop additive-E | additive-E**

**=>提取左公因子 simple-E → additive-E simple-E’**

**simple-E’-> relop additive-E | epsilon**

21. relop → <= | < | > | >= | = = | != | && | ||

**22. additive-E → additive-E aop term | term**

**=>消除左递归 additive-E-> term additive-E’**

**additive-E’-> aop term additive-E’ | epsilon**

23. aop → + | -

24. term → term mop F | F

**=>消除左递归 term->F term’**

**term’ -> mop F term’ | epsilon**

25. mop → \* | /

26. F → (E) | V | call | NUM | DECIMAL

27. call → ID (As)

**=>19 26 27 三个式子合并，提取左公因子**

**F →ID F’ | (E) | NUM | DECIMAL**

**F’ →[E] | epsilon| (As)**

28. As → A-L | empty

**29. A-L → A-L, E | E**

**=>消除左递归 A-L->E A-L’**

**A-L’-> , E A-L’ | epsilon**

**（二）消除左递归和左公因子后的文法 （这里为节省篇幅，含有 | 的产生式不分开写）**

1. PR → D-L
2. **D-L →D D-L | epsilon**
3. **D → T-S ID D‘**
4. **D‘ → epsilon | [NUM] | (Ps) C-S**
5. T-S → int | void | float
6. Ps → P-L | **epsilon**
7. **P-L→P P-L‘**
8. **P-L‘ →, P P-L‘ | epsilon**
9. P → T-S **ID**
10. C-S → { local-D S-L }
11. **local-D → V-D local-D | empty**
12. **S-L → S S-L | empty**
13. S → E-S | C-S | selection-S | iteration-S | return-S
14. E-S → E;|;
15. **selection-S → if (E) S selection-S’**
16. **selection-S’ → else S | epsilon**
17. iteration-S → while (E) S
18. **return-S → return return-S’**
19. **return-S’ → ; | E;**
20. E → V = E | simple-E
21. **V → ID V’**
22. **V’-> [E] | epsilon**
23. **simple-E → additive-E simple-E’**
24. **simple-E’-> relop additive-E | epsilon**
25. relop → <= | < | > | >= | = = | != | && | ||
26. **additive-E-> term additive-E’**
27. **additive-E’-> aop term additive-E’ | epsilon**
28. aop → + | -
29. **term->F term’**
30. **term’ -> mop F term’ | epsilon**
31. mop → \* | /
32. **F →ID F’ | (E) | NUM | DECIMAL**
33. **F’ →[E] | epsilon| (As)**
34. As → A-L | empty
35. **A-L->E A-L’**
36. **A-L’-> , E A-L’ | epsilon**

第二步，求出first 集 follow 集

**FIRST集\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**First(PR)={ int void float epsilon }**

**First(D-L)= { int void float epsilon }**

**First(D)= { int void float }**

**FIRST(D’)= { [ ( epsilon }**

**First(V-D)= { int void float }**

**First(V-D’)={ [ epsilon }**

**First(T-S)= { int void float }**

**First(P)= { int void float }**

**FIRST(Ps)= { int void float epsilon }**

**FIRST(P-L)= { int void float }**

**FIRST(P-L’)= { , epsilon }**

**FIRST(C-S)={ { }**

**FIRST(local-D)= { int void float epsilon }**

**FIRST(S-L)={ ; { if while return ( ID NUM DECIMAL epsilon }**

**FIRST(S)={ ; { if while return ( ID NUM DECIMAL }**

**FIRST(E-S)={ ; ( ID NUM DECIMAL }**

**FIRST(selection-S)={ if }**

**FIRST(selection-S’)={ else epsilon }**

**FIRST(Iteration-S)={ while }**

**FIRST(return-S)={ return }**

**FIRST(return-S’)={ ; ( ID NUM DECIMAL }**

**FIRST(E)= { ( ID NUM DECIMAL }**

**FIRST(V)={ ID }**

**FIRST(V’)={ [ epsilon }**

**FIRST(simple-E)= { ( ID NUM DECIMAL }**

**FIRST(simple-E’)={ < > <= >= == != && || epsilon }**

**FIRST(additive-E)= { ( ID NUM DECIMAL }**

**FIRST(additive-E’)={ + - epsilon }**

**FIRST(term)= { ( ID NUM DECIMAL }**

**FIRST(term’)={ \* / epsilon }**

**FIRST(F)=FIRST(term)=FIRST(additive-E)=FIRST(simple-E)= { ( ID NUM DECIMAL }**

**FIRST(F’)={ [ ( EPSILON }**

**FIRST(As)={ ( ID NUM DECIMAL epsilon }**

**FIRST(A-L)= { ( ID NUM DECIMAL }**

**FIRST(A-L’)={ , epsilon }**

**FIRST(relop)= { < > <= >= == != && || }**

**FIRST(aop)={ + - }a**

**FIRST(mop)={ \* / }**

**FOLLOW集合\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**FOLLOW(PR)={ # }**

**FOLLOW(D-L)= { # }**

**FOLLOW(D)= { int void float # }**

**FOLLOW(D’)= { int void float # }**

**FOLLOW(V-D)= { int void float ; { if while return ( ID NUM DECIMAL }**

**FOLLOW(V-D’)={ int void float ; { if while return ( ID NUM DECIMAL }**

**FOLLOW(T-S)= { ID }**

**FOLLOW(Ps)={ ) }**

**FOLLOW(P)= { , ) }**

**FOLLOW(P-L)= { ) }**

**FOLLOW(P-L’)= { ) }**

**FOLLOW(C-S)={ int void float # else } ; { if while return ( ID NUM DECIMAL }**

**FOLLOW(local-D)= { ; { if while return ( ID NUM DECIMAL }**

**FOLLOW(S-L)= { } }**

**FOLLOW(S)={ else } ; { if while return ( ID NUM DECIMAL }**

**FOLLOW(E-S)={ else } ; { if while return ( ID NUM DECIMAL }**

**FOLLOW(selection-S)= { else } ; { if while return ( ID NUM DECIMAL }**

**FOLLOW(selection-S’)= { else } ; { if while return ( ID NUM DECIMAL }**

**FOLLOW(Iteration-S)= { else } ; { if while return ( ID NUM DECIMAL }**

**FOLLOW(return-S)= { else } ; { if while return ( ID NUM DECIMAL }**

**FOLLOW(return-S’)= { else } ; { if while return ( ID NUM DECIMAL }**

**FOLLOW(E)={ ) ] } , else ; }**

**FOLLOW(V)={ = }**

**FOLLOW(V’)={ = }**

**FOLLOW(simple-E)={ ; ) ] , }**

**FOLLOW(simple-E’)={ ; ) ] , }**

**FOLLOW(additive-E)={ ; ) ] , < > <= >= == != && || }**

**FOLLOW(additive-E’)={ ; ) ] , < > <= >= == != && || }**

**FOLLOW(term)={ + - ; ) ] , < > <= >= == != && || }**

**FOLLOW(term’)={ + - ; ) ] , < > <= >= == != && || }**

**FOLLOW(F)={ \* / + - ; ) ] , < > <= >= == != && || }**

**FOLLOW(F’)={ \* / + - ; ) ] , < > <= >= == != && || }**

**FOLLOW(As)={ ) }**

**FOLLOW(A-L)={ ) }**

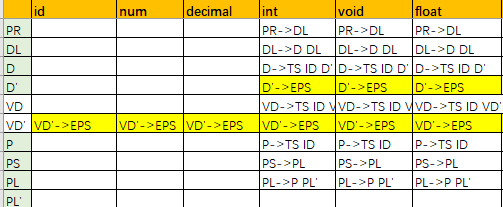
**FOLLOW(A-L’)={ ) }**

**FOLLOW(relop)= { ( ID NUM DECIMAL }**

**FOLLOW(aop)={ ( ID NUM DECIMAL }**

**FOLLOW(mop)={ ( ID NUM DECIMAL }**

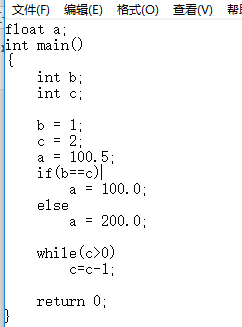
第三步，构建预测分析表



完整版见附录1

## 测试结果

**（一）测试代码如下：**



**（二）用第一次试验的词法分析器，分析代码，得到结果。**

对第一次词法分析程序做如下改进：

若为语法分析中定义的终结符，则输出终结符，否则输出类型。输出结果如下：

int

id

(

)

{

int

id

;

int

id

;

id

=

num

;

id

=

num

;

id

=

decimal

;

if

(

id

==

id

)

id

=

decimal

;

else

id

=

decimal

;

while

(

id

>

num

)

id

=

id

-

num

;

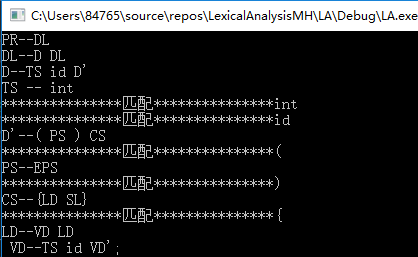
return

num

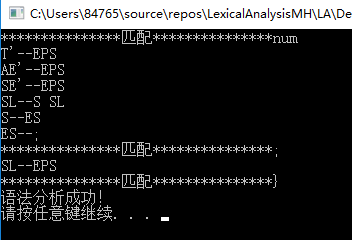
;

}

**（三）将此结果作为语法分析器的输入得到产生式输出结果**



中间省去若干行……



**（四）构建语法树**

在输出分析结果的同时输出dot语言。输出的 dot 语言 如下

(完整版见压缩包文件 Expression.txt)

digraph g{

node0[label="PR"]

node101[label="DL"]

node0->node101

node201[label="D"]

node101->node201

node102[label="DL"]

……

node3813[label="id"]

node2910->node3813

node3007[label="F'"]

node2910->node3007

node6837[label="eps"]

node3007->node6837

node6838[label="eps"]

node2810->node6838

node3501[label="AOP"]

node2610->node3501

node2711[label="T"]

node2610->node2711

node2611[label="AE'"]

node2610->node2611

node2911[label="F"]

node2711->node2911

node2811[label="T'"]

node2711->node2811

node3904[label="num"]

node2911->node3904

node6839[label="eps"]

node2811->node6839

node6840[label="eps"]

node2611->node6840

node6841[label="eps"]

node2408->node6841

node1309[label="S"]

node1206->node1309

node1207[label="SL"]

node1206->node1207

node1801[label="RS"]

node1309->node1801

node4701[label="return"]

node1801->node4701

node1901[label="RS'"]

node1801->node1901

node2015[label="E"]

node1901->node2015

node2309[label="SE"]

node2015->node2309

node2511[label="AE"]

node2309->node2511

node2409[label="SE'"]

node2309->node2409

node2712[label="T"]

node2511->node2712

node2612[label="AE'"]

node2511->node2612

node2912[label="F"]

node2712->node2912

node2812[label="T'"]

node2712->node2812

node3905[label="num"]

node2912->node3905

node6842[label="eps"]

node2812->node6842

node6843[label="eps"]

node2612->node6843

node6844[label="eps"]

node2409->node6844

node1310[label="S"]

node1207->node1310

node1208[label="SL"]

node1207->node1208

node1407[label="ES"]

node1310->node1407

node4809[label=";"]

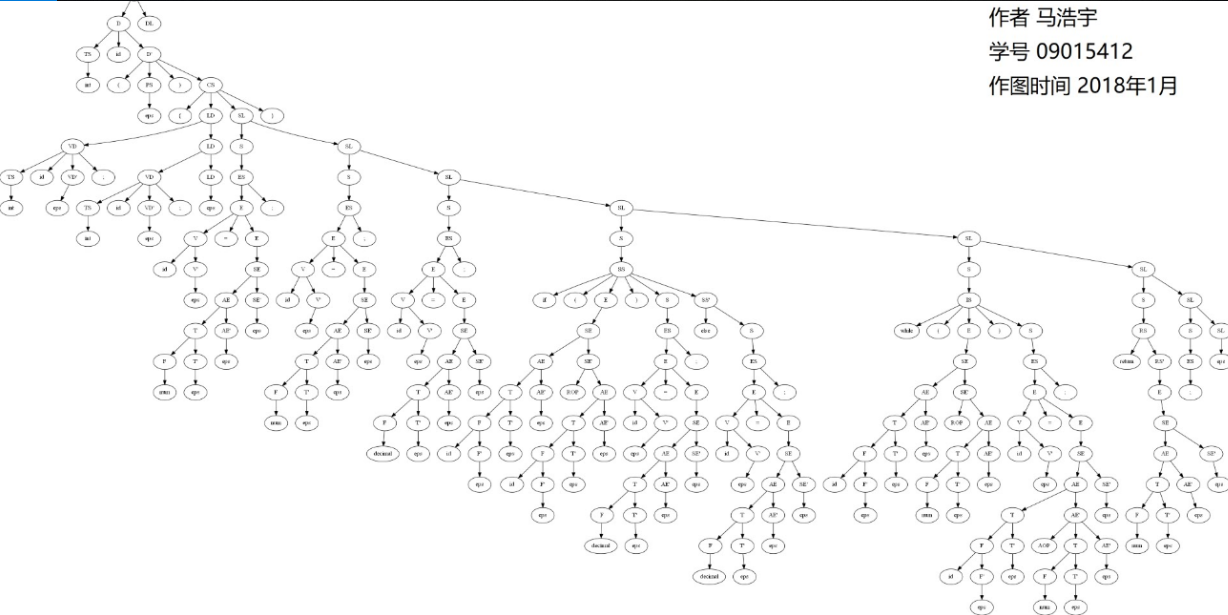
node1407->node4809

node6845[label="eps"]

node1208->node6845

}

采用graphviz作图,语法树如下（大图见附录2）



## 源码

#include "stdio.h"

#include "string"

#include<stdlib.h>

#include<iostream>

#include<fstream>

#include<stack>

#include<queue>

#include <map>

using namespace std;

stack<string>downstack;//下推栈

queue<string>inputstr;//输入字符串

ofstream fout;//

//非终结符

string VN[38] = {

"PR","DL","D","D'","VD","VD'","P","PS","PL","PL'",

"CS","LD","SL","S","ES","SS","SS'","IS","RS","RS'"

,"E","V","V'","SE","SE'","AE","AE'","T","T'","F",

"F'","AS","AL","AL'","ROP","AOP","MOP","TS"};

//终结符

string VT[31] = {"id","num","decimal", "int","void","float","if","else","while","return",

";",",","(",")","[","]","{","}","=","+","-","\*","/","<",">","<=",">=","==","!=","&&","||" };

stack<int>VNCOUNT[69];//用于统计输出图每个节点的标号

int vcount[69] = { 0 };

void printLink(int n1, int n2, string nodeName1, string nodeName2)

{

int t1, t2;

t1 = VNCOUNT[n1].top();//左边用栈顶原节点值

t2=++vcount[n2];//右边用新建的节点值

VNCOUNT[n2].push(t2);

fout << "node" << t2 << "[label=\"" << nodeName2 << "\"]" << endl;

fout << "node" << t1 << "->" << "node" << t2 << endl;

}

bool SearchTable(string X, string a);

//判断是否为终结符

bool IsVT(string str) {

for (int i = 0; i < 29; i++) {

if (VT[i]==str)

return true;

}

return false;

}

//判断是否为非终结符

bool ISVN(string str) {

for (int i = 0; i < 38; i++) {

if (VN[i] == str)

return true;

}

return false;

}

//初始化输入串队列

void QueueInitialize(FILE \*inf) {

char ch;

string str="";

while (!feof(inf)) {

ch = fgetc(inf);

if (ch != '\n')

str = str + ch;

else {

inputstr.push(str);

str = "";

}

}

inputstr.push("#");

for (int i = 0; i < 69; i++) {

VNCOUNT[i].push( i \* 100);//初始化Node VN编号

vcount[i] = i \* 100;

}

}

//词法分析

bool GrammarAnalysis() {

downstack.push("#");//

downstack.push("PR");//

while (inputstr.front() != "#") {//输入串没读完

string X = downstack.top();//下推栈栈顶

string a = inputstr.front();//输入串串首

if (X == a){//匹配

downstack.pop();//弹出栈顶符号X

inputstr.pop();//读头前进一格

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*匹配\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << X << endl;

continue;

}

if (IsVT(X) && (X != a)) {//如果X属于VT X不等于a

cout << "语法分析出错！" << endl;

return false;//ERROR

}

if (ISVN(X)) {

if (SearchTable(X, a)) {

}

else {

cout << "语法分析出错！" << endl;

return false;//ERROR

}

}

else {

cout << "语法分析出错！" << endl;

return false;

}

}

cout << "语法分析成功！" << endl;

return true;

}

//查询LL1表

bool SearchTable(string X,string a) {

//E--SE E--V=E 由于语法本身问题，故此处往后多看一个字符

if (X == "E" &&a == "id") {

int len = inputstr.size();//当前输入字符串长度

inputstr.pop();//弹出队列头

inputstr.push(a);//队列头放入队列尾

string a2 = inputstr.front();//队列第二个元素

int i = 0;

string buf = "";

while (i < len - 1) {

buf = inputstr.front();//队列头

inputstr.pop();//弹出

inputstr.push(buf);//放入队列尾

i++;

}//恢复原队列

if (a2 == "=") {//E-- V = E

downstack.pop();

downstack.push("E");

downstack.push("=");

downstack.push("V");

printLink(20, 21, "E", "V");

printLink(20, 56, "E", "=");

printLink(20, 20, "E", "E");

//VNCOUNT[20].pop();//原始值用后要释放 递归调用

}

else { //E--SE

downstack.pop();

downstack.push("SE");

printLink(20, 23, "E", "SE");

VNCOUNT[20].pop();//原始值用后要释放

}

return true;

}

if (X == "PR" && (a == "int" || a == "void" || a == "float" || a == "#")) {

downstack.pop();

downstack.push("DL");

cout << "PR--DL" << endl;

//输出graphviz格式的语言

printLink(0, 1,"PR" , "DL" );

VNCOUNT[0].pop();//原始值用后要释放

return true;

}

if (X == "DL" && (a == "int" || a == "void" || a == "float" || a == "#")) {

downstack.pop();//DL出栈

downstack.push("DL");//产生式自右向左压栈

downstack.push("D");

cout << "DL--D DL" << endl;

//输出graphviz格式的语言

printLink(1, 2, "DL", "D");

printLink(1, 1, "DL", "DL");

// VNCOUNT[1].pop();//原始值用后要释放 递归调用

return true;

}

if (X == "DL" && a == "#") {

downstack.pop();//DL出栈 此时产生式是空串

cout << "DL--EPS" << endl;

printLink(1, 68, "DL", "eps");

VNCOUNT[1].pop();//原始值用后要释放

return true;

}

if (X == "D" && (a == "int" || a == "void" || a == "float")) {

downstack.pop();//DL出栈

downstack.push("D'");//产生式自右向左压栈

downstack.push("id");

downstack.push("TS");

cout << "D--TS id D'" << endl;

//输出graphviz格式的语言

printLink(2, 37, "D", "TS");

printLink(2, 38, "D", "id");

printLink(2, 3, "D", "D'");

VNCOUNT[2].pop();//原始值用后要释放

return true;

}

if (X == "D'" && (a == "int" || a == "void" || a == "float")) {//D'--EPS

downstack.pop();//D'出栈

cout << "D'--EPS" << endl;

printLink(3, 68, "D'", "eps");

VNCOUNT[3].pop();//原始值用后要释放

return true;

}

if (X == "D'" && (a == "[")) {//D'--[ NUM ]

downstack.pop();//D'出栈

downstack.push("]");//产生式自右向左压栈

downstack.push("num");

downstack.push("[");

cout << "D'--[ num ]" << endl;

//输出graphviz格式的语言

printLink(3, 50, "D'", "[");

printLink(3, 39, "D'", "num");

printLink(3, 51, "D'", "]'");

VNCOUNT[3].pop();//原始值用后要释放

return true;

}

if (X == "D'" && (a == "(")) {//D'--(PS)CS

downstack.pop();//D'出栈

downstack.push("CS");//产生式自右向左压栈

downstack.push(")");

downstack.push("PS");

downstack.push("(");

cout << "D'--( PS ) CS" << endl;

//输出graphviz格式的语言

printLink(3, 52, "D'", "(");

printLink(3, 7, "D'", "PS");

printLink(3, 53, "D'", ")");

printLink(3, 10, "D'", "CS");

VNCOUNT[3].pop();//原始值用后要释放

return true;

}

if (X == "VD" && (a == "int" || a == "void" || a == "float")) {//VD--TS ID VD'

downstack.pop();//D'出栈

downstack.push(";");

downstack.push("VD'");//产生式自右向左压栈

downstack.push("id");

downstack.push("TS");

cout << " VD--TS id VD';" << endl;

printLink(4, 37, "VD", "TS");

printLink(4, 38, "VD", "id");

printLink(4, 5, "VD", "VD'");

printLink(4, 48, "VD", ";");

VNCOUNT[4].pop();//原始值用后要释放

return true;

}

if (X == "VD'" && (a == "[")) {//VD'--[ NUM ]

downstack.pop();//VD'出栈

downstack.push("]");//产生式自右向左压栈

downstack.push("num");

downstack.push("[");

cout << " VD'--[ num ]" << endl;

//输出graphviz格式的语言

printLink(5, 50, "VD'", "[");

printLink(5, 39, "VD'", "num");

printLink(5, 51, "VD'", "]");

VNCOUNT[5].pop();//原始值用后要释放

return true;

}

if (X == "VD'" && (a == "int" || a == "void" || a == "float"

||a=="if"||a=="while" ||a=="return"

||a=="id" ||a=="num"||a=="decimal"

||a==";")) {

downstack.pop();//VD'出栈

cout << " VD'--EPS" << endl;

printLink(5, 68, "VD'", "eps");

VNCOUNT[5].pop();//原始值用后要释放

return true;

}

//P--TS ID

if (X == "P" && (a == "int" || a == "void" || a == "float")) {

downstack.pop();//VD'出栈

downstack.push("id");//产生式自右向左压栈

downstack.push("TS");

cout << " P--TS id" << endl;

//输出graphviz格式的语言

printLink(6, 37, "P", "TS");

printLink(6, 38, "P", "id");

VNCOUNT[6].pop();//原始值用后要释放

return true;

}

//PS--PL

if (X == "PS" && (a == "int" || a == "void" || a == "float")) {

downstack.pop();//VD'出栈

downstack.push("PL");//产生式自右向左压栈

cout << "PS--PL" << endl;

//输出graphviz格式的语言

printLink(7, 8, "PS", "PL");

VNCOUNT[7].pop();//原始值用后要释放

return true;

}

if (X == "PS" && (a ==")")) {//Ps--EPS

downstack.pop();//VD'出栈

cout << "PS--EPS" << endl;

printLink(7, 68, "PS", "eps");

VNCOUNT[7].pop();//原始值用后要释放

return true;

}

//PL--P PL'

if (X == "PL" && (a == "int" || a == "void" || a == "float")) {

downstack.pop();//VD'出栈

downstack.push("PL'");//产生式自右向左压栈

downstack.push("P");

cout << "PL--P PL'" << endl;

//输出graphviz格式的语言

printLink(8, 6, "PL", "P");

printLink(8, 9, "PL", "PL'");

VNCOUNT[8].pop();//原始值用后要释放

return true;

}

//PL'--, P PL'

if (X == "PL'" && (a == ",")) {

downstack.pop();//VD'出栈

downstack.push("PL'");//产生式自右向左压栈

downstack.push("P");

downstack.push(",");

cout << "PL'--, P PL'" << endl;

//输出graphviz格式的语言

printLink(9, 49, "PL'", ",");

printLink(9, 6, "PL'", "P");

printLink(9, 9, "PL'", "PL'");

//VNCOUNT[9].pop();//原始值用后要释放 递归调用

return true;

}

if (X == "PL'" && (a == ")")) {//PL'--EPS

downstack.pop();//VD'出栈

cout << "PL'--EPS'" << endl;

printLink(9, 68, "PL'", "eps");

VNCOUNT[9].pop();//原始值用后要释放

return true;

}

if (X == "CS" && (a == "{")) {//CS--{LD SL}

downstack.pop();//VD'出栈

downstack.push("}");

downstack.push("SL");

downstack.push("LD");

downstack.push("{");

cout << "CS--{LD SL}" << endl;

//输出graphviz格式的语言

printLink(10, 54, "CS", "{");

printLink(10, 11, "CS", "LD");

printLink(10, 12, "CS", "SL");

printLink(10, 55, "CS", "}");

VNCOUNT[10].pop();//原始值用后要释放

return true;

}

//LD--VD LD

if (X == "LD" && (a == "int" || a == "void" || a == "float")) {

downstack.pop();//VD'出栈

downstack.push("LD");//产生式自右向左压栈

downstack.push("VD");

cout << "LD--VD LD" << endl;

printLink(11, 4, "LD", "VD");

printLink(11, 11, "LD", "LD");

//VNCOUNT[11].pop();//原始值用后要释放 递归调用

return true;

}

//LD--EPS

if (X == "LD" && (a == ";" || a == "("||a=="{"

|| a == "if" || a == "while" || a == "return"

|| a == "id" || a == "num" || a == "decimal"

)) {

downstack.pop();//VD'出栈

cout << "LD--EPS" << endl;

printLink(11, 68, "LD", "eps");

VNCOUNT[11].pop();//原始值用后要释放

return true;

}

//SL--S SL

if (X == "SL" && (a == ";" || a == "(" || a == "{"

|| a == "if" || a == "while" || a == "return"

|| a == "id" || a == "num" || a == "decimal"

)) {

downstack.pop();

downstack.push("SL");

downstack.push("S");

cout << "SL--S SL" << endl;

printLink(12, 13, "SL", "S");

printLink(12, 12, "SL", "SL");

return true;

}

//SL--EPS

if (X == "SL" && (a =="}")) {

downstack.pop();//VD'出栈

cout << "SL--EPS" << endl;

printLink(12, 68, "SL", "eps");

VNCOUNT[12].pop();//原始值用后要释放

return true;

}

//S--ES FIRST(ES)={ ; ( ID NUM DECIMAL }

if (X == "S" && (a == "id" || a == "num" || a == "decimal"

|| a == ";" || a == "(" )) {

downstack.pop();

downstack.push("ES");

cout << "S--ES" << endl;

printLink(13, 14, "S", "ES");

VNCOUNT[13].pop();//原始值用后要释放

return true;

}

//S--SS

if (X == "S" && (a == "if")) {

downstack.pop();

downstack.push("SS");

cout << "S--SS" << endl;

printLink(13, 15, "S", "SS");VNCOUNT[13].pop();//原始值用后要释放

return true;

}

//S--IS

if (X == "S" && (a == "while")) {

downstack.pop();

downstack.push("IS");

cout << "S--IS" << endl;

printLink(13, 17, "S", "IS"); VNCOUNT[13].pop();//原始值用后要释放

return true;

}

//S--RS

if (X == "S" && (a == "return")) {

downstack.pop();

downstack.push("RS");

cout << "S--RS" << endl;

printLink(13, 18, "S", "RS"); VNCOUNT[13].pop();//原始值用后要释放

return true;

}

//S--CS

if (X == "S" && (a == "{")) {

downstack.pop();

downstack.push("CS");

cout << "S--CS" << endl;

printLink(13, 10, "S", "CS"); VNCOUNT[13].pop();//原始值用后要释放

return true;

}

//ES--E; FIRST(E)= { ( ID NUM DECIMAL }

if (X == "ES" && (a == "{" || a == "id" || a == "num" || a == "decimal") ) {

downstack.pop();

downstack.push(";");

downstack.push("E");

cout << "ES--E;" << endl;

printLink(14, 20, "ES", "E");

printLink(14, 48, "ES", ";");

VNCOUNT[14].pop();//原始值用后要释放

return true;

}

//ES--;

if (X == "ES" && (a == ";")) {

downstack.pop();

downstack.push(";");

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

printLink(14, 48, "ES", ";"); VNCOUNT[14].pop();//原始值用后要释放

return true;

}

//SS--if ( E ) S SS'

if (X == "SS" && (a == "if")) {

downstack.pop();

downstack.push("SS'");

downstack.push("S");

downstack.push(")");

downstack.push("E");

downstack.push("(");

downstack.push("if");

cout << "SS-- if ( E ) S SS'" << endl;

printLink(15, 44, "SS", "if");

printLink(15, 52, "SS", "(");

printLink(15, 20, "SS", "E");

printLink(15, 53, "SS", ")");

printLink(15, 13, "SS", "S");

printLink(15, 16, "SS", "SS'"); VNCOUNT[15].pop();//原始值用后要释放

return true;

}

//SS' -- else S

if (X == "SS'" && (a == "else")) {

downstack.pop();

downstack.push("S");

downstack.push("else");

cout << "SS' -- else S " << endl;

printLink(16, 45, "SS'", "else");

printLink(16, 13, "SS'", "S"); VNCOUNT[16].pop();//原始值用后要释放

return true;

}

//SS'--EPS FOLLOW(SS’)= { else } ; { ( if while return ID NUM DECIMAL }

if (X == "SS'" && (a == ";" || a == "{" ||a=="}"|| a=="("

|| a == "if" || a == "while" || a == "return"

|| a == "id" || a == "num" || a == "decimal"

)) {

downstack.pop();

cout << "SS'--EPS" << endl;

printLink(16, 68, "SS'", "eps");

VNCOUNT[16].pop();//原始值用后要释放

return true;

}

//IS--while( E ) S

if (X == "IS" && (a == "while")) {

downstack.pop();

downstack.push("S");

downstack.push(")");

downstack.push("E");

downstack.push("(");

downstack.push("while");

cout << "IS-- while ( E ) S" << endl;

printLink(17, 46, "IS", "while");

printLink(17, 52, "IS", "(");

printLink(17, 20, "IS", "E");

printLink(17, 53, "IS", ")");

printLink(17, 13, "IS", "S"); VNCOUNT[17].pop();//原始值用后要释放

return true;

}

//RS--return RS'

if (X == "RS" && (a == "return")) {

downstack.pop();

downstack.push("RS'");

downstack.push("return");

cout << "RS--return RS'" << endl;

printLink(18, 47, "RS", "return");

printLink(18, 19, "RS", "RS'"); VNCOUNT[18].pop();//原始值用后要释放

return true;

}

//RS'--E FIRST(E)= { ( ID NUM DECIMAL }

if (X == "RS'" && (a == "{" || a == "id" || a == "num" || a == "decimal")) {

downstack.pop();

downstack.push("E");

cout << "RS'--E" << endl;

printLink(19,20 , "RS'", "E"); VNCOUNT[19].pop();//原始值用后要释放

return true;

}

//RS'--;

if (X == "RS'" && (a ==";")) {

downstack.pop();

downstack.push(";");

cout << "RS'--;" << endl;

printLink(19, 48, "RS'", ";"); VNCOUNT[19].pop();//原始值用后要释放

return true;

}

//E--SE FIRST(simple-E)= { ( ID NUM DECIMAL }

if (X == "E" && (a == "(" || a == "num" || a == "decimal")) {

downstack.pop();

downstack.push("SE");

cout << "E--SE" << endl;

printLink(20, 23, "E", "SE"); VNCOUNT[20].pop();//原始值用后要释放

return true;

}

//V--id V'

if (X == "V" && (a == "id")) {

downstack.pop();

downstack.push("V'");

downstack.push("id");

cout << "V--id V'" << endl;

printLink(21, 38, "V", "id");

printLink(21, 22, "V", "V'"); VNCOUNT[21].pop();//原始值用后要释放

return true;

}

//V'--[E]

if (X == "V'" && (a == "[")) {

downstack.pop();

downstack.push("]");

downstack.push("E");

downstack.push("[");

cout << "V'--[E]" << endl;

printLink(22,50 , "V'", "[");

printLink(22, 20, "V'", "E");

printLink(22, 51, "V'", "]"); VNCOUNT[22].pop();//原始值用后要释放

return true;

}

//V'--EPS

if (X == "V'" && (a == "=")) {

downstack.pop();

cout << "V'--EPS" << endl;

printLink(22, 68, "V'", "eps");

VNCOUNT[22].pop();//原始值用后要释放

return true;

}

//SE--AE SE' FIRST(additive-E)= { ( ID NUM DECIMAL }

if (X == "SE" && (a == "(" || a == "id" || a == "num" || a == "decimal")) {

downstack.pop();

downstack.push("SE'");

downstack.push("AE");

cout << "SE--AE SE'" << endl;

printLink(23, 25, "SE", "AE");

printLink(23, 24, "SE", "SE'"); VNCOUNT[23].pop();//原始值用后要释放

return true;

}

//SE'--EPS FOLLOW(simple-E’)={ ; ) ] , }

if (X == "SE'" && (a == ";" || a == ")" || a == "]" || a == ",")) {

downstack.pop();

cout << "SE'--EPS" << endl;

printLink(24, 68, "SE'", "eps");

VNCOUNT[24].pop();//原始值用后要释放

return true;

}

//SE'--ROP AE FIRST(ROP)= { < > <= >= == != && || }

if (X == "SE'" && (a == "<" || a == ">" || a == "<=" || a == ">="

|| a == "==" || a == "!=" || a == "&&"|| a == "||"

)) {

downstack.pop();

downstack.push("AE");

downstack.push("ROP");

cout << "SE'--ROP AE" << endl;

printLink(24, 34, "SE'", "ROP");

printLink(24, 25, "SE'", "AE"); VNCOUNT[24].pop();//原始值用后要释放

return true;

}

//AE--T AE' FIRST(T)= { ( ID NUM DECIMAL }

if (X == "AE" && (a == "(" || a == "id" || a == "num" || a == "decimal")) {

downstack.pop();

downstack.push("AE'");

downstack.push("T");

cout << "AE--T AE'" << endl;

printLink(25, 27, "AE", "T");

printLink(25, 26, "AE", "AE'"); VNCOUNT[25].pop();//原始值用后要释放

return true;

}

//AE'--EPS FOLLOW(AE’)={ ; ) ] , < > <= >= == != && || }

if (X == "AE'" && (a == "<" || a == ">" || a == "<=" || a == ">="

|| a == "==" || a == "!=" || a == "&&" || a == "||"

|| a == ";" || a == ")" || a == "]" || a == ","

)) {

downstack.pop();

cout << "AE'--EPS" << endl;

printLink(26, 68, "AE'", "eps");

VNCOUNT[26].pop();//原始值用后要释放

return true;

}

//AE'--AOP T AE'

if (X == "AE'" && (a == "+" || a == "-" )) {

downstack.pop();

downstack.push("AE'");

downstack.push("T");

downstack.push("AOP");

cout << "AE'--AOP T AE'" << endl;

printLink(26, 35, "AE'", "AOP");

printLink(26, 27, "AE'", "T");

printLink(26, 26, "AE'", "AE'");

//VNCOUNT[26].pop();//原始值用后要释放 递归调用

return true;

}

//T--FT' FIRST(F)= { ( ID NUM DECIMAL }

if (X == "T" && (a == "(" || a == "id" || a == "num" || a == "decimal") ) {

downstack.pop();

downstack.push("T'");

downstack.push("F");

cout << "T-- F T'" << endl;

printLink(27, 29, "T", "F");

printLink(27, 28, "T", "T'"); VNCOUNT[27].pop();//原始值用后要释放

return true;

}

//T'--MOP FT'

if (X == "T'" && (a == "\*" || a == "/" )) {

downstack.pop();

downstack.push("T'");

downstack.push("F");

downstack.push("MOP");

cout << "T'--MOP F T'" << endl;

printLink(28, 36, "T'", "MOP");

printLink(28, 29, "T'", "F");

printLink(28, 28, "T'", "T'");

//VNCOUNT[28].pop();//原始值用后要释放

return true;

}

//T'--EPS FOLLOW(T’)={ + - ; ) ] , < > <= >= == != && || }

if (X == "T'" && (a == "<" || a == ">" || a == "<=" || a == ">="

|| a == "==" || a == "!=" || a == "&&" || a == "||"

|| a == ";" || a == ")" || a == "]" || a == ","

|| a == "+" || a == "-"

)) {

downstack.pop();

cout << "T'--EPS" << endl;

printLink(28, 68, "T'", "eps");

VNCOUNT[28].pop();//原始值用后要释放

return true;

}

//F--id F'

if (X == "F" && (a == "id")) {

downstack.pop();

downstack.push("F'");

downstack.push("id");

cout << "F--id F'" << endl;

printLink(29, 38, "F", "id");

printLink(29, 30, "F", "F'"); VNCOUNT[29].pop();//原始值用后要释放

return true;

}

//F--NUM

if (X == "F" && (a == "num")) {

downstack.pop();

downstack.push("num");

cout << "F--num" << endl;

printLink(29, 39, "F", "num"); VNCOUNT[29].pop();//原始值用后要释放

return true;

}

//F--DECIMAL

if (X == "F" && (a == "decimal")) {

downstack.pop();

downstack.push("decimal");

cout << "F--decimal" << endl;

printLink(29, 30, "F", "decimal"); VNCOUNT[29].pop();//原始值用后要释放

return true;

}

//F--( E )

if (X == "F" && (a == "(")) {

downstack.pop();

downstack.push(")");

downstack.push("E");

downstack.push("(");

cout << "F--( E )" << endl;

printLink(29, 52, "F", "(");

printLink(29, 20, "F", "E");

printLink(29, 53, "F", ")"); VNCOUNT[29].pop();//原始值用后要释放

return true;

}

// F'--EPS FOLLOW(F’)={ \* / + - ; ) ] , < > <= >= == != && || }

if (X == "F'" && (a == "<" || a == ">" || a == "<=" || a == ">="

|| a == "==" || a == "!=" || a == "&&" || a == "||"

|| a == ";" || a == ")" || a == "]" || a == ","

|| a == "+" || a == "-" || a=="\*" || a=="/"

)) {

downstack.pop();

cout << "F'--EPS" << endl;

printLink(30, 68, "F'", "eps");

VNCOUNT[30].pop();//原始值用后要释放

return true;

}

//F'-- [ E ]

if (X == "F'" && (a == "[")) {

downstack.pop();

downstack.push("]");

downstack.push("E");

downstack.push("[");

cout << "F'-- [ E ]" << endl;

printLink(30, 50, "F'", "[");

printLink(30, 20, "F'", "E");

printLink(30, 51, "F'", "]"); VNCOUNT[30].pop();//原始值用后要释放

return true;

}

//F'--(As)

if (X == "F'" && (a == "(")) {

downstack.pop();

downstack.push(")");

downstack.push("AS");

downstack.push("(");

cout << "F'-- ( AS )" << endl;

printLink(30, 52, "F'", "(");

printLink(30, 31, "F'", "AS");

printLink(30, 53, "F'", ")"); VNCOUNT[30].pop();//原始值用后要释放

return true;

}

//As--AL FIRST(A-L)= { ( ID NUM DECIMAL }

if (X == "AS" && (a == "(" || a == "id" || a == "num" || a == "decimal")) {

downstack.pop();

downstack.push("AL");

cout << "AS--AL" << endl;

printLink(31, 32, "AS", "AL"); VNCOUNT[31].pop();//原始值用后要释放

return true;

}

//As--EPS

if (X == "AS" && (a == ")")) {

downstack.pop();

cout << "AS--EPS" << endl;

printLink(31, 68, "AS", "eps");

VNCOUNT[31].pop();//原始值用后要释放

return true;

}

//AL--E AL' FIRST(E)= { ( ID NUM DECIMAL }

if (X == "AL" && (a == "(" || a == "id" || a == "num" || a == "decimal")) {

downstack.pop();

downstack.push("AL'");

downstack.push("E");

cout << "AL--E AL'" << endl;

printLink(32, 20, "AL", "E");

printLink(32, 33, "AL", "AL'"); VNCOUNT[32].pop();//原始值用后要释放

return true;

}

//AL'-- , E AL'

if (X == "AL'" && (a == ",")) {

downstack.pop();

downstack.push("AL'");

downstack.push("E");

downstack.push(",");

cout << "AL'-- , E AL' " << endl;

printLink(33, 49, "AL'", ",");

printLink(33, 20, "AL'", "E");

printLink(33, 33, "AL'", "AL'");

return true;

}

//AL'--EPS

if (X == "AL'" && (a == ")")) {

downstack.pop();

cout << "AL'-- EPS" << endl;

printLink(33, 68, "AL'", "eps");

VNCOUNT[33].pop();//原始值用后要释放

return true;

}

//ROP

if (X == "ROP") {

downstack.pop();

if(a=="<")

downstack.push("<");

if (a == ">")

downstack.push(">");

if (a == "<=")

downstack.push("<=");

if (a == ">=")

downstack.push(">=");

if (a == "==") {

downstack.push("==");

}

if (a == "!=")

downstack.push("!=");

if (a == "&&")

downstack.push("&&");

if (a == "||")

downstack.push("||");

cout << "ROP" << endl;

//fout << "ROP" << endl;

return true;

}

//MOP

if (X == "MOP") {

downstack.pop();

if (a == "\*") {

downstack.push("\*");

cout << "MOP -- \*" << endl;

//fout << "MOP -- \*" << endl;

}

if (a == "/") {

downstack.push("/");

cout << "MOP -- /" << endl;

//fout << "MOP -- /" << endl;

}

return true;

}

//AOP

if (X == "AOP") {

downstack.pop();

if (a == "+") {

downstack.push("+");

cout << "AOP -- +" << endl;

//fout << "AOP -- +" << endl;

}

if (a == "-") {

downstack.push("-");

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

//fout << "AOP -- -" << endl;

}

return true;

}

//TS

if (X == "TS") {

downstack.pop();

if (a == "int"){

downstack.push("int");

cout << "TS -- int" << endl;

printLink(37, 41, "TS", "int"); VNCOUNT[37].pop();//原始值用后要释放

//fout << "TS -- int" << endl;

}

if (a == "void") {

downstack.push("void");

cout << "TS -- void" << endl;

printLink(37, 42, "TS", "void"); VNCOUNT[37].pop();//原始值用后要释放

//fout << "TS -- void" << endl;

}

if (a == "float") {

downstack.push("float");

cout << "TS -- float" << endl;

printLink(37, 43, "TS", "float"); VNCOUNT[37].pop();//原始值用后要释放

//fout << "TS -- float" << endl;

}

return true;

}

return false;

}

int main()

{

FILE \* inputfile;

if ((inputfile = fopen("LA.txt", "r")) == NULL) {

cout << "File doesn't exist!" << endl;

system("pause");

return 0;

}

else {

fout.open("Expression.txt");

fout << "digraph g{" << endl;

fout << "node" << 0 << "[label=\"" << "PR" << "\"]" << endl;

QueueInitialize(inputfile);//初始化输入串队列

GrammarAnalysis();//词法分析

fout << "}" << endl;

fclose(inputfile);

fout.close();

}

system("pause");

return 0;

}