**1.C-语言的语法图描述**

S -> program

program -> declaration-list

declaration-list -> declaration-list declaration | declaration

declaration -> var-declaration | fun-declaration

var-declaration -> type-specifier ID ; | type-specifier ID [ NUM ] ;

type-specifier -> int | void

fun-declaration -> type-specifier IDF ( params ) | compound-stmt

params -> params-list | void

params-list -> params-list , param | param

param -> type-specifier ID | type-specifier ID [ ]

compound-stmt -> { local-declarations statement-list }

local-declarations -> local-declarations var-declaration | empty

statement-list -> statement-list statement | empty

statement -> expression-stmt | compound-stmt | selection-stmt | iteration-stmt | return-stmt

expression-stmt -> expression ; | ;

selection-stmt -> if ( expression ) statement | if ( expression ) statement else statement

iteration-stmt -> while ( expression ) statement

return-stmt -> return ; | return expression ;

expression -> var1 = expression | simple-expression

var -> ID | ID [ expression ]

var1 -> ID1 | ID1 [ expression ]

simple-expression -> additive-expression relop additive-expression | additive-expression

relop -> <= | < | > | >= | == | !=

additive-expression -> additive-expression addop term | term

addop -> + | -

term -> term mulop factor | factor

mulop -> \* | /

factor -> ( expression ) | var | call | NUM

call -> IDF ( args )

args -> arg-list | empty

arg-list -> arg-list , expression | expression

**2.系统设计**

由0 prog-resource.txt存储源程序，4 grammar.txt存储语法规则

程序读入两个文件后，首先进行预处理操作，对源程序去除注释，去除多余空格，对语法规则消除左递归以及公共左因子，并且根据|拆分语法推导式，根据后序语法分析发现的问题后回过来对语法做的一些细微的修改也在这一步完成。

随后对处理过后的源程序进行词法分析，得到一个tokne流，供语法分析时使用，生成栈以及语法分析树。

对处理后的语法规则，首先计算出所有的终结符以及非终结符，并求出非终结符的first,follow集，生成预测分析表。

根据token流计算出符号表

根据token流以及预测分析表完成整个程序的出入栈以及语法树的建立

使用深度遍历语法树遇到相应节点，即进行语义动作，产生三地址代码。

最后对产生额三地址代码进行优化。

**3.系统实现**

**3.1系统主要函数说明（主要功能、输入\输出、实现思想）**

(所有函数以及变量定义见附word文档)

* main函数中调用各步骤操作

void precedure**(**string period**,** int num**,** string filein**,** string fileout**,** void**(\***func**)(**fstream **&,** fstream **&),**string content**);**

获取第一、二个参数为当前过程名称及过程中的第几步，第三、四个参数获取获取的文件名以及输出的目标文件名，第五个参数为进行该步操作所需要调用的函数名称，分别定义在了各分块的文件中，最后一个参数为该步骤所进行为何操作。

函数中首先打开了输入输出文件并判断是否打开成功，随后调用获取的目标函数，将两个文件流操作变量传递给它。输出当前步骤的各种信息以及各文件名作为提示。函数最后将两个文件都关闭。

函数中调用了打开输入文件函数infile,打开输出文件函数outfile,关闭文件函数closefile以及输出分割线函数divideline.

* Pretreat中

void delzs**(**fstream **&**in**,** fstream **&**out**);**

void delblank**(**fstream **&**in**,** fstream **&**out**);**

函数主体均由switch-case语句构成，来实现有限状态自动机，详细的状态转换见pretreat.h，消除多余空白符时，会记录前一个出现的空白字符，如果为回车则会永久记录下来，保证尽量不改变代码换行的位置。去除注释时，在判断一段文字是否为注释时，使用一个字符串记录还未被确定不是注释的内容，以防在文件末尾注释右侧未封闭导致出现代码的缺失(虽然这种情况下是显然不符合语法规则的代码)。

3.pretreat中

void delleftrecursion**(**fstream **&**in**,** fstream **&**out**);**

函数用于消除语法的左递归以及公共左因子，其中首先将一个推导式以字符串读入，根据->的位置将字符串进行拆分，记录左侧字符串为part1，右侧为part2。下一步将part2根据|拆分为多个部分，该操作调用函数splitblank，返回一个数组其中存储了各部分的字符串值和一个int值表示有几个部分。下一步由于观察语法推导式可以发现所有推导式出现左递归的情况只会出现在右侧第一个推导，所以只判断第一部分是否出现了左递归，如果出现了则按照A->Ab|c A->cA’和A’->bA’|empty。随后判断公共左因子，也很容易发现存在公共左因子的情况也只会出现在右侧只有两个部分时，那么也可以简化算法，判断前两个是否出现了公共的左因子，也按照类似左递归进行消除。

* pretreat中

void splitgrammar**(**fstream **&**in**,** fstream **&**out**)**

函数中将已经适应ll1规则的语法规则每一条根据|进行拆分，并在每一条推导式前添加编号进行计数。

* grammar-analysis中

各类初始化函数

void init\_vttype**(**map**<**string**,** string**>** **&**vttype**)**

初始化终结符类型表使用string-string类型的map表，键为原始终结符符号，值为其类型，具体分类见grammar-analysis.h中定义。

void init\_mtable**(**struct Mtable **&**mtable**)**

初始化预测分析表使用专用定义的结构体，初始化时将预测分析表的第一列填充为所有非终结符，第一行填充为所有终结符，并且所有位置预设初始值0即不能推导。

void stack\_init**(**struct Stack **&**stack**)**

初始化栈时首先压入最终的#以及所有推导式的第一个产生的非终结符，即默认程序一开始已经读入一个#并且出栈(默认起始入栈#[firststr]#)

void tree\_init**()**

初始化语法分析树，首先构建根节点treenode，给它三个分支# [firststr] #，将所有节点信息预设赋值。

void init\_funcmap**()**

初始化语义动作表，每一条语法产生式都对应着一个语义动作函数，以键值对$[num]-fomula\_[num]定义，有特殊的推导式例如if以及while产生式需要多个不同位置的语义动作，则添加多个定义。

* grammar-analysis

void cal\_vn\_vt**(**fstream **&**in**,**fstream **&**out**)**

该函数计算出所有的非终结符以及终结符，函数中逐条读入推导式，根绝->将推导式分为左右两个部分，对于左边部分，判断是否已经出现在非终结符表中，如果没有出现则加入，并判断是否出现在终结符表中，如果出现则从终结符表中删除，对于右边部分，判断是否出现在终结符表中，如果未出现则加入。使用check\_exsist传入字符串以及表明，返回1表示出现，0表示未出现，使用del传入表明以及字符串删除表中对应的字符串。

* grammar-analysis中

void cal\_first**()**

该函数计算所有非终结符的first集，整体使用while(1)循环，使用一个flag，如果整个循环中对于所有的first集合没有进行改变则跳出循环。

First、follow集均采用string-vector<string>存储，键为非终结符名称，值为存储first或follow字符串的vector容器

* 循环初始flag=1 逐条分析推导式 推导式使用int-vector<string>的map类型存储，int表示推导式的编号，vector<string>存储根据空格拆分后推导式。
* 判断推导式第一个元素是否为终结符没如果为终结符则判断该终结符是否在当前非终结符的first集中，如果不在则加入并且flag=0
* 如果第一个元素为非终结符，则将该非终结符的first集中的所有元素加入推导式首部的非终结符的first集中，且flag=0
* 如果推导式前部分的非终结符的first集中有empty，则将他们所有的first集中的所有元素加入至首部的非终结符的first集中，且flag=0，直到遇到一个不能推导出empty的非终结符或是终结符。
* grammar-analysis中

void cal\_follow**()**

该函数计算所有非终结符的follow集，整体使用while(1)循环，使用一个flag，如果整个循环中对于所有的follow集合没有进行改变则跳出循环。

* 循环初始flag=1 逐条分析推导式 最开始将#置于[firststr]的follow集中 分析推导式时逐个遍历式中元素
* 当前元素为终结符，无需操作，进入下一个元素
* 当前元素为非终结符，向后遍历直到出现终结符或非终结符不能推导出empty时停止，将所有能推出empty的非终结符的first集中所有元素或终结符加入该非终结符的follow集中
* 由产生式的最后向前遍历，直到遇到终结符或不能推导empty的非终结符停止将推导式首部的非终结符的follow集添加至尾部所有能推导出empty的非终结符的follow集中
* grammar-analysis中

void cal\_mtable**(**fstream **&**in**,** fstream **&**out**)**

该函数计算出预测分析表，遍历推导式表，对于每一条推导式，先取出箭头左侧的非终结符作为预测分析表的行，再取出箭头右侧第一个符号，对于它的所有的first集，将该推导式编号填入first集内容所对应的列中，如果遇到empty将emptyflag赋值为1，循环后，判断推导式左侧的非终结符是否能退出空，如果有，则将那条推导式编号填入该非终结符follow集元素所对应的列中。

* grammar-analysis中

void make\_tree**(**fstream **&**in**,** fstream **&**out**)**

该函数在对token流读取并根据预测分析表进行出入栈的同时建立语法分析树。

* 出入栈操作

当栈顶元素为终结符时，将栈顶元素出栈并判断当前读进的符号是否与出栈元素相同，如果不相同则报错。

当栈顶元素为非终结符时，将此非终结符作为行，读入的符号作为列，查看预测分析表中该行该列的值，如果为0则报错，如果不为零，则先将栈顶元素出栈，然后将其值所对应的推导式由后往前一次入栈

* 建树操作

首先设置当前节点为treenode根节点的son[2]位置。

每次使用推导式推导时，将推导式所有元素按顺序新建节点插入当前节点的sons中，并在推导结束时添加$[num]的语义操作节点。

每次推导结束时进入当前节点的sons[1]节点，每次出栈时进入当前节点的brother节点，如果当前节点无brother节点，则进入其father节点后再次寻找brother节点，知道brother节点存在并且不为语义动作节点。

* grammar-analysis中

bool fomula\_[num] **(**treebit **\***temp**)**

各语义动作节点

* grammar-analysis中

void midcode**(**fstream **&**in**,** fstream **&**out**)**

该函数负责中间代码生成，使用深度优先遍历遍历整个语法分析树，调用deepsearch函数，碰到语义动作节点则调用相对应的语义动作函数。

* symbol中

定义变量结构体和函数结构体

struct symbol

{

string name; //变量名

string type; //变量类型

string property; //变量属性

struct symbol \*st; //函数型成员地址

}; //变量元素结构体

struct func

{

string name; //函数名

string ret; //函数返回值

int num; //函数中参数个数

struct symbol st[255]; //函数符号表

}; //函数结构体

* symbol中

int func\_exist(struct func functable[255], int funcnum, string str)

判断当前函数是否存在符号表中，若存在返回该函数在符号表中的位置否则返回-1

int sym\_exist(struct func functable[255], struct symbol \*st, int num, string str)

判断当前变量时候存在符号表中，若存在返回该变量在符号表中的位置否则返回-1

* symbol中

void pushsym(struct func functable[255], struct symbol \*st, int curnum, int num)

将当前函数名加入到当前函数表中

void pushsym2(struct func functable[255], struct symbol \*st, int curnum, int num, string name, string property, string type)

将当前变量名加入到当前函数表中

* symbol中

struct symbol \* pushfunc(struct func functable[255], int &funcnum, string str, string ret)

将当前函数加入到函数表中，并返回该函数符号表的地址

* symbol中

void init\_symtable(void)

初始化符号表

void output(fstream &out)

输出符号表

* symbol中

void calsymboltable(fstream &in, fstream &out)

生成符号表

* 读入词法分析中的每一个token
* 遍历所有的token，在遍历的过程中检测当前token属性并作出相应操作。

在检测过程中设置了两个flag用于标记大括号和小括号的计数，从而判断当前token所处的位置，再结合对应的属性对token进行分类处理。

* 如果当前token为函数名。函数分为定义和调用的两种情况。

如果该函数名token之前不存在左大括号，说明是函数的定义，检查表中是否有该函数，若无该函数则将该函数写入表中。

如果该函数名之前存在一个或多个左大括号，则说明这是函数的调用。检查符号表，若表中不存在该函数说明未进行函数声明就调用，进入报错处理；若存在该函数再判断该函数名中的变量是否存在表中，如果是首次调用则将变量加入到函数表中。

* 如果当前token为变量名，同样分为变量定义和调用的处理。

如果该变量名前有‘int’或‘void’，则表示是变量的定义，而变量的定义只能存在函数参数的小括号内或者不能存在小括号内部，根据这一条件判断是否合法。若是合法的变量定义再检查表中，若不存在则入表。

如果变量是调用，检查表中是否存在，若不存在，说明变量未经定义就调用，进入报错处理。

**3.2系统代码（必须有注释）**

* symbol

#include "symbol.h"

struct symbol

{

string name; //变量名

string type; //变量类型

string property; //变量属性

struct symbol \*st; //函数型成员地址

}; //变量元素结构体

struct func

{

string name; //函数名

string ret; //函数返回值

int num; //函数中参数个数

struct symbol st[255]; //函数符号表

}; //函数结构体

struct func functable[255]; //符号表

int funcnum = 0; //符号表中元素个数

//判断函数是否已经存在 并返回在该函数在函数名表中的位置 不存在则返回-1

int func\_exist(struct func functable[255], int funcnum, string str)

{

int i;

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

if (functable[i].name == str)

return i;

return -1;

}

//判断变量是否出现在当前函数的符号表中 存在则返回表中位置 否则返回-1

int sym\_exist(struct func functable[255], struct symbol \*st, int num, string str)

{

int i;

for (i = 0;i < functable[num].num;i++)

if ((st + i)->name == str)

return i;

return -1;

}

//将当前函数名加入到当前函数表中

void pushsym(struct func functable[255], struct symbol \*st, int curnum, int num)

{

st[functable[curnum].num].name = functable[num].name;

st[functable[curnum].num].property = "IDF";

st[functable[curnum].num].type = "-";

st[functable[curnum].num].st = functable[num].st;

functable[curnum].num++;

}

//将当前变量加入到当前函数表中

void pushsym2(struct func functable[255], struct symbol \*st, int curnum, int num, string name, string property, string type)

{

st[functable[curnum].num].name = name;

st[functable[curnum].num].property = property;

st[functable[curnum].num].type = type;

functable[curnum].num++;

}

//将当前函数加入到函数表中 并返回该函数符号表的地址

struct symbol \* pushfunc(struct func functable[255], int &funcnum, string str, string ret)

{

functable[funcnum].name = str;

functable[funcnum].num = 0;

functable[funcnum].ret = ret;

funcnum++;

return functable[funcnum - 1].st;

}

//初始化符号表

void init\_symtable(void)

{

pushfunc(functable, funcnum, "input", "-");

pushfunc(functable, funcnum, "output", "-");

}

//输出符号表

void output(fstream &out)

{

int i, j;

int set = 20;

//输出函数符号表

out << setw(set) << left << "函数名" << setw(set) << "函数中参数个数" << setw(set) << "函数返回类型" << setw(set) << "函数符号表地址" << endl;

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

out << setw(set - 1) << left << functable[i].name << setw(set - 2) << functable[i].num << setw(set - 2) << functable[i].ret << "0x" << setw(set) << functable[i].st << endl;

//输处各函数表

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

{

out << endl;

out << "函数名:" << functable[i].name << "\t符号表地址:0x" << functable[i].st << endl;

if (functable[i].num)

out << setw(set) << "变量名" << setw(set) << "属性" << setw(set) << "类型" << setw(set) << "地址" << endl;

for (j = 0;j < functable[i].num;j++)

out << setw(set - 1) << functable[i].st[j].name << setw(set - 1) << functable[i].st[j].property << setw(set - 1) << functable[i].st[j].type << setw(set) << functable[i].st[j].st << endl;

}

}

//生成符号表

void calsymboltable(fstream &in, fstream &out)

{

vector<Token> tokens;

Token token;

vector<Token>::iterator titer;

struct symbol \*cur = NULL;

int flag1, flag2, num, curnum;

//读入词法分析结果

while (!in.eof())

{

in >> token.re >> token.type >> token.line >> token.colume;

if (in.fail())

break;

tokens.push\_back(token);

}

flag1 = 0;

flag2 = 0;

init\_symtable();

//检测当前token的属性为函数名或者变量名，根据标记括号的flag1.flag2判断位置

for (titer = tokens.begin();titer != tokens.end();titer++)

{

if (titer->type == "IDF") //当前token为函数名,分为函数的定义和调用情况

{

num = func\_exist(functable, funcnum, titer->re);

if (flag1 == 0)//之前不存在左大括号，说明当前为函数的定义

{

if (num != -1)//已存在该函数名

{

cur = functable[num].st; //得到该函数名在符号表中的信息

curnum = num;

}

else //不存在该函数名,写入该函数名

{

cur = pushfunc(functable, funcnum, titer->re, (titer - 1)->re);

curnum = funcnum - 1;

}

}

else if (flag1>0) //说明是函数内部对函数的调用

{

if (num == -1) //函数符号表中未出现 函数没有被定义就被调用 报错

{

cout << "Undefined Function [" << titer->re << "] at Line " << titer->line << " Colume " << titer->colume << endl;

exit(0);

}

else//已存在该函数名，判断是否为首次调用 是则加入至当前函数的变量表中

{

if (sym\_exist(functable, cur, curnum, titer->re) == -1)

pushsym(functable, cur, curnum, num);

}

}

}

else if (titer->type == "ID") //当前token为变量名，分为变量的声明和调用情况

{

if ((titer - 1)->re == "void" || (titer - 1)->re == "int") //前一个token为int或void，说明当前为变量的声明

{

if (flag1 == 0 || (flag1 > 0 && flag2 == 0)) //变量声明只能在函数获取形参中或不在其他位于函数内部的小括号内

{

if (sym\_exist(functable, cur, curnum, titer->re) == -1) //不存在该变量，则定义，入表

pushsym2(functable, cur, curnum, num, titer->re, "DEFINE", (titer - 1)->re);

else //存在该变量，不用入表

{

cout << "Variable [" << titer->re << "] has been defined!" << endl;

exit(0);

}

}

else

{

cout << "//变量定义在错误的位置 ";

}

}

else //前一个token不是int或void，说明为函数的调用

{

if (sym\_exist(functable, cur, curnum, titer->re) == -1) //符号表中不存在，说明未定义就使用

{

cout << "Undefined Variable [" << titer->re << "] at Line " << titer->line << " Colume " << titer->colume << endl;

exit(0);

}

/\* else //符号表中存在

{

cout << "Variable [" << titer->re << "] has been defined!" << endl;

cout << titer->line;

exit(0);

}\*/

}

}

else if (titer->re == "{")

flag1++;

else if (titer->re == "}")

flag1--;

else if (titer->re == "(")

flag2++;

else if (titer->re == ")")

flag2--;

}

output(out);

}

* gramma-analysis
* #include "grammar-analysis.h"
* Token token**;**
* struct Mtable mtable**;**
* struct Stack stack**;**
* treebit **\***treenode**;**
* int nodenum **=** 0**;** //节点编号
* map**<**int**,** vector**<**string**>>** placelistmap**;** //各节点存储字符串信息表
* map**<**string**,** bool**(\*)(**treebit **\***cur**)>** funcmap**;** //推导式编号与其对应的语义动作函数
* map**<**int**,** string**>** midcodemap**;**
* map**<**string**,** int**>** funcnum**;**
* int curnum **=** 99**;** //三地址代码编号
* int curtemp **=** 0**;** //临时变量个数
* //初始化终结符类型表
* void init\_vttype**(**map**<**string**,** string**>** **&**vttype**)**
* **{**
* vttype**.**insert**({** "<"**,**"COP" **});**
* vttype**.**insert**({** "<="**,**"COP" **});**
* vttype**.**insert**({** ">"**,**"COP" **});**
* vttype**.**insert**({** ">="**,**"COP" **});**
* vttype**.**insert**({** "=="**,**"COP" **});**
* vttype**.**insert**({** "!="**,**"COP" **});**
* vttype**.**insert**({** "="**,**"AOP" **});**
* vttype**.**insert**({** "+"**,**"OOP" **});**
* vttype**.**insert**({** "-"**,**"OOP" **});**
* vttype**.**insert**({** "\*"**,**"OOP" **});**
* vttype**.**insert**({** "/"**,**"OOP" **});**
* vttype**.**insert**({** ";"**,**"EOP" **});**
* vttype**.**insert**({** ","**,**"SOP" **});**
* vttype**.**insert**({** "("**,**"SOP" **});**
* vttype**.**insert**({** ")"**,**"SOP" **});**
* vttype**.**insert**({** "["**,**"SOP" **});**
* vttype**.**insert**({** "]"**,**"SOP" **});**
* vttype**.**insert**({** "{"**,**"SOP" **});**
* vttype**.**insert**({** "}"**,**"SOP" **});**
* vttype**.**insert**({** "int"**,**"REVERSED" **});**
* vttype**.**insert**({** "if"**,**"REVERSED" **});**
* vttype**.**insert**({** "else"**,**"REVERSED" **});**
* vttype**.**insert**({** "return"**,**"REVERSED" **});**
* vttype**.**insert**({** "void"**,**"REVERSED" **});**
* vttype**.**insert**({** "while"**,**"REVERSED" **});**
* vttype**.**insert**({** "NUM"**,**"NUM" **});**
* vttype**.**insert**({** "ID"**,**"ID" **});**
* vttype**.**insert**({** "IDF"**,**"IDF" **});**
* vttype**.**insert**({** "ID1"**,**"ID1" **});**
* vttype**.**insert**({** "empty"**,**"empty" **});**
* vttype**.**insert**({** "#"**,**"#" **});**
* **}**
* //初始化预测分析表mtable
* void init\_mtable**(**struct Mtable **&**mtable**)**
* **{**
* int i**,** j**,** vnnum**,** vtnum**;**
* **for** **(**iter **=** vn**.**begin**(),** vnnum **=** 1**;**iter **!=** vn**.**end**();**iter**++,** vnnum**++)**
* mtable**.**vnname**.**insert**({** **\***iter**,**vnnum **});**
* //预测分析表中的终结符部分需要去除empty 产生空字的列
* **for** **(**iter **=** vt**.**begin**(),** vtnum **=** 1**;**iter **!=** vt**.**end**();**iter**++,** vtnum**++)**
* **if** **(\***iter **!=** "empty"**)**
* mtable**.**vtname**.**insert**({** **\***iter**,**vtnum **});**
* **else**
* vtnum**--;**
* //初始化全赋值为0 推导式编号由1开始
* **for** **(**i **=** 1**;**i **<** vnnum**;**i**++)**
* **for** **(**j **=** 1**;**j **<** vtnum**;**j**++)**
* mtable**.**mtable**[**i**][**j**]** **=** 0**;**
* mtable**.**vnnum **=** vnnum **-** 1**;**
* mtable**.**vtnum **=** vtnum **-** 1**;**
* **}**
* //判断集合中是否存在str元素
* int check\_exist**(**vector**<**string**>** vec**,** string str**)**
* **{**
* vector**<**string**>::**iterator iter**;**
* iter **=** find**(**vec**.**begin**(),** vec**.**end**(),** str**);**
* **if** **(**iter **!=** vec**.**end**())**
* **return** 1**;**
* **else**
* **return** 0**;**
* **}**
* //判断vtmap中是否有该vt
* int checkmap**(**map**<**string**,** string**>** mp**,** string str**)**
* **{**
* iter\_map **=** vttype**.**find**(**str**);**
* **if** **(**iter\_map **!=** vttype**.**end**())**
* **return** 1**;**
* **else**
* **return** 0**;**
* **}**
* //删除vt集中应该为vn的元素
* void del**(**vector**<**string**>** **&**vec**,** string str**)**
* **{**
* iter **=** find**(**vec**.**begin**(),** vec**.**end**(),** str**);**
* vec**.**erase**(**iter**);**
* **}**
* //计算终结符集合以及非终结符集合
* void cal\_vn\_vt**(**fstream **&**in**,**fstream **&**out**)**
* **{**
* string temp**,**part1**,**part2**,**t**;**
* int split**,** start**,** index**,** num **=** 1**;**
* init\_vttype**(**vttype**);**
* **while** **(!**in**.**eof**())**
* **{**
* getline**(**in**,** temp**);**
* **if** **(**in**.**fail**())**
* **break;**
* temp**.**replace**(**0**,** temp**.**find**(**' '**),** ""**);** //去掉每行开头的数字
* split **=** temp**.**find**(**"->"**);**
* part1**.**assign**(**temp**,** 0**,** split**);**
* part2**.**assign**(**temp**,** split **+** 2**,** temp**.**length**()** **-** split **-** 1**);**
* fix**(**part1**);**
* fix**(**part2**);** //将每一行语法分为箭头前后的两个部分
* start **=** 0**;**
* grammarmap**.**insert**({** num**,{**part1**}** **});**
* //左侧部分 如果vn集合中还没有出现则加入 如果vt集合中出现则将vt集合中的该元素删除
* **if** **(!**check\_exist**(**vn**,** part1**))**
* vn**.**push\_back**(**part1**);**
* **if** **(**check\_exist**(**vt**,** part1**))**
* del**(**vt**,** part1**);**
* **while** **(**1**)** //将箭头后的部分按照空格分开
* **{**
* index **=** part2**.**find**(**' '**,**start**);**
* **if** **(**index **==** part2**.**npos**)**
* **break;**
* t**.**assign**(**part2**,** start**,** index **-** start**);**
* start **=** index **+** 1**;**
* //当右侧部分在 vn中没有出现且vt中没有出现 则将该元素加入vt集合中
* **if** **(!**check\_exist**(**vn**,** t**)** **&&** **!**check\_exist**(**vt**,** t**))**
* vt**.**push\_back**(**t**);**
* grammarmap**[**num**].**push\_back**(**t**);**
* **}**
* t**.**assign**(**part2**,** start**,** part2**.**length**()** **-** start**);**
* **if** **(!**check\_exist**(**vn**,** t**)** **&&** **!**check\_exist**(**vt**,** t**))**
* vt**.**push\_back**(**t**);**
* //填充可以推导出空的非终结符表
* grammarmap**[**num**].**push\_back**(**t**);**
* **if** **(**grammarmap**[**num**][**1**]** **==** "empty"**)**
* emptymake**.**insert**({** grammarmap**[**num**][**0**],**num **});**
* num**++;**
* **}**
* vt**.**push\_back**(**"#"**);**
* //输出
* output\_vec**(**out**);**
* **}**
* //输出vector中所有信息 输出vn,vt信息至文件
* void output\_vec**(**fstream **&**out**)**
* **{**
* out **<<** "非终结符vn:" **<<** endl**;**
* **for** **(**iter **=** vn**.**begin**();**iter **!=** vn**.**end**();**iter**++)**
* **{**
* out **<<** **\***iter **<<** endl**;**
* firstlist**.**insert**({** **\***iter**,** **{}** **});**
* followlist**.**insert**({** **\***iter**,{}** **});**
* **if** **(\***iter **==** firststr**)**
* followlist**[**firststr**].**push\_back**(**"#"**);**
* **}**
* out **<<** "终结符vt:" **<<** endl**;**
* **for** **(**iter **=** vt**.**begin**();**iter **!=** vt**.**end**();**iter**++)**
* **{**
* **if** **(!**checkmap**(**vttype**,** **\***iter**))**
* **{**
* //cout << "Undefined vt[ " << \*iter << " ]!" << endl;
* //exit(0);
* **;**
* **}**
* out **<<** **\***iter **<<** endl**;**
* firstlist**.**insert**({** **\***iter**,** **{\***iter**}** **});**
* **}**
* **}**
* //求first集
* void cal\_first**()**
* **{**
* int flag**,** outflag**;**
* unsigned int k**;**
* string vnname**,** tname**,** temp**;**
* **while** **(**1**)**
* **{**
* flag **=** 1**;**
* **for** **(**gmaper **=** grammarmap**.**begin**();**gmaper **!=** grammarmap**.**end**();**gmaper**++)**
* **{**
* vnname **=** gmaper**->**second**[**0**];**
* **if** **(**check\_exist**(**vt**,** gmaper**->**second**[**1**]))**
* **{**
* **if** **(!**check\_exist**(**firstlist**[**vnname**],** gmaper**->**second**[**1**]))**
* **{**
* firstlist**[**vnname**].**push\_back**(**gmaper**->**second**[**1**]);**
* flag **=** 0**;**
* **}** //判断该终结符是否已经在该推导的first集中,没有则添加
* **}** //产生式第一个元素为终结符
* **else**
* **{**
* k **=** 1**;**
* outflag **=** 1**;**
* **while** **(**1**)**
* **{**
* **if** **(**k **>=** gmaper**->**second**.**size**())**
* **break;**
* //cout << k << " " << gmaper->second.size() << endl;
* tname **=** gmaper**->**second**[**k**];**
* **if** **(**check\_exist**(**vn**,** tname**))**
* **{**
* //逐个检验是否已经在vnname的first集中,不存在就添加
* **for** **(**iter **=** firstlist**[**tname**].**begin**();**iter **!=** firstlist**[**tname**].**end**();**iter**++)**
* **{**
* temp **=** **\***iter**;**
* **if** **(!**check\_exist**(**firstlist**[**vnname**],** temp**))**
* **{**
* firstlist**[**vnname**].**push\_back**(**temp**);**
* flag **=** 0**;**
* **}**
* **if** **(\***iter **==** "empty"**)** //如果该非终结符的first集中没有empty则进入下一推导式
* outflag **=** 0**;**
* **}**
* **}**
* **else**
* **{** //推导式中出现终结符 判断是否添加后 进入下一个推导式
* outflag **=** 1**;**
* **if** **(!**check\_exist**(**firstlist**[**vnname**],** tname**))**
* **{**
* firstlist**[**vnname**].**push\_back**(**tname**);**
* flag **=** 0**;**
* **}**
* **}**
* **if** **(**outflag**)**
* **break;**
* k**++;**
* **}**
* **}**
* **}**
* //当flag为1时表示已经没有first集发生添加元素 fisrt集合求解完成
* **if** **(**flag**)**
* **break;**
* **}**
* **}**
* //求follow集
* void cal\_follow**()**
* **{**
* int flag**,** outflag**,** exflag**;**
* string vnname**,** tname**;**
* vector**<**string**>::**iterator titer**,** temp**;**
* **while** **(**1**)**
* **{**
* flag **=** 1**;**
* **for** **(**gmaper **=** grammarmap**.**begin**();**gmaper **!=** grammarmap**.**end**();**gmaper**++)** //遍历推导式
* **{**
* **for** **(**iter **=** gmaper**->**second**.**begin**()** **+** 1**;**iter **!=** gmaper**->**second**.**end**();**iter**=**titer**)** //遍历推导式中个符号
* **{**
* titer **=** iter **+** 1**;**
* **if** **(**check\_exist**(**vt**,** **\***iter**))** //遇到终结符 不需要计算follow集
* **continue;**
* **else**
* **{**
* vnname **=** **\***iter**;** //遇到一个非终结符开始向后遍历 知道遇到不能推导空的非终结符或是终结符
* **while** **(**1**)**
* **{**
* iter**++;**
* outflag **=** 1**;** //记录一个非终结符能不能推导出空
* **if** **(**iter **==** gmaper**->**second**.**end**())** //到推导式结束 当前非终结符follow集操作完成
* **break;**
* **if** **(**check\_exist**(**vt**,** **\***iter**))**
* **{**
* **if** **(!**check\_exist**(**followlist**[**vnname**],** **\***iter**)** **&&** **\***iter **!=** "empty"**)**
* **{**
* followlist**[**vnname**].**push\_back**(\***iter**);**
* flag **=** 0**;**
* **}**
* **}** //遇到一个终结符 判断是否存在 不存在则加入当前非终结符的follow集中
* **else**
* **{**
* **for** **(**temp **=** firstlist**[\***iter**].**begin**();**temp **!=** firstlist**[\***iter**].**end**();**temp**++)**
* **{**
* **if** **(!**check\_exist**(**followlist**[**vnname**],** **\***temp**)** **&&** **\***temp **!=** "empty"**)**
* **{**
* followlist**[**vnname**].**push\_back**(\***temp**);**
* flag **=** 0**;**
* **}**
* **if** **(\***temp **==** "empty"**)**
* outflag **=** 0**;**
* **}** //将当前符号后遇到的所有的first集合判断并加入到follow集中 直到遇到不能推导出空的非终结符
* **}**
* **if** **(**outflag**)**
* **break;**
* **}**
* **}**
* **}**
* unsigned int i **=** 1**;**
* //推导式最后连续i个非终结符可以推导为空 则推导式最前的非终结符的follow集中所有元素判断并添加到这所有的非终结符的follow集中
* **while** **(**1**)**
* **{**
* **if** **(**check\_exist**(**vn**,** **\*(**gmaper**->**second**.**end**()** **-** i**)))**
* **{**
* string str**;**
* **if** **(**i **>=** gmaper**->**second**.**size**())**
* **break;**
* str **=** **\*(**gmaper**->**second**.**end**()** **-** i**);**
* vnname **=** gmaper**->**second**[**0**];**
* **for** **(**temp **=** followlist**[**vnname**].**begin**();**temp **!=** followlist**[**vnname**].**end**();**temp**++)**
* **{**
* **if** **(!**check\_exist**(**followlist**[**str**],** **\***temp**)** **&&** **\***temp **!=** "empty"**)**
* **{**
* followlist**[**str**].**push\_back**(\***temp**);**
* flag **=** 0**;**
* **}**
* **}**
* exflag **=** 0**;**
* **for** **(**iter **=** firstlist**[**str**].**begin**();**iter **!=** firstlist**[**str**].**end**();**iter**++)**
* **if** **(\***iter **==** "empty"**)**
* exflag **=** 1**;**
* //exflag记录该非终结符的first集中是否有empty
* **if** **(!**exflag**)**
* **break;**
* **else**
* i**++;**
* **}**
* **else**
* **break;**
* **}**
* **}**
* **if** **(**flag**)**
* **break;**
* **}**
* **}**
* //计算first follow集
* void cal\_first\_follow**(**fstream **&**in**,** fstream **&**out**)**
* **{**
* cal\_first**();**
* cal\_follow**();**
* output\_ff**(**out**);**
* **}**
* //输出first,follow集至文件
* void output\_ff**(**fstream **&**out**)**
* **{**
* //输出first集
* out **<<** "//first集" **<<** endl**;**
* **for** **(**maper **=** firstlist**.**begin**();**maper **!=** firstlist**.**end**();**maper**++)**
* **{**
* **if** **(!**check\_exist**(**vt**,** maper**->**first**))**
* **{**
* out **<<** maper**->**first **<<** "的first集:"**;**
* **for** **(**iter **=** maper**->**second**.**begin**();**iter **!=** maper**->**second**.**end**();**iter**++)**
* out **<<** **\***iter **<<** " "**;**
* out **<<** endl**;**
* **}**
* **}**
* //输出follow集
* out **<<** "//follow集" **<<** endl**;**
* **for** **(**maper **=** followlist**.**begin**();**maper **!=** followlist**.**end**();**maper**++)**
* **{**
* **if** **(!**check\_exist**(**vt**,** maper**->**first**))**
* **{**
* out **<<** maper**->**first **<<** "的follow集:"**;**
* **for** **(**iter **=** maper**->**second**.**begin**();**iter **!=** maper**->**second**.**end**();**iter**++)**
* out **<<** **\***iter **<<** " "**;**
* out **<<** endl**;**
* **}**
* **}**
* **}**
* //计算预测分析表
* void cal\_mtable**(**fstream **&**in**,** fstream **&**out**)**
* **{**
* string name**,** vnname**;**
* int emptyflag**,** i**,** j**;**
* init\_mtable**(**mtable**);**
* //首先输出终结符非终结符个数
* out **<<** "vnnum: " **<<** mtable**.**vnnum **<<** endl**;**
* out **<<** "vtnum: " **<<** mtable**.**vtnum **<<** endl**;**
* **for** **(**gmaper **=** grammarmap**.**begin**();**gmaper **!=** grammarmap**.**end**();**gmaper**++)**
* **{**
* name **=** gmaper**->**second**[**1**];** //当前推导式箭头右侧第一个符号
* vnname **=** gmaper**->**second**[**0**];** //当前推导式箭头左侧的非终结符
* emptyflag **=** 0**;**
* **for** **(**iter **=** firstlist**[**name**].**begin**();**iter **!=** firstlist**[**name**].**end**();**iter**++)**
* **{**
* **if** **(\***iter **==** "empty"**)**
* **{**
* emptyflag **=** 1**;**
* **continue;**
* **}**
* //将name的所有first集所对应的位置填入当前推导式的编号
* mtable**.**mtable**[**mtable**.**vnname**[**vnname**]][**mtable**.**vtname**[\***iter**]]** **=** gmaper**->**first**;**
* **}**
* **if** **(**name **==** "empty"**&&**gmaper**->**second**.**size**()** **>** 2**)**
* **{**
* name **=** gmaper**->**second**[**2**];**
* **for** **(**iter **=** firstlist**[**name**].**begin**();**iter **!=** firstlist**[**name**].**end**();**iter**++)**
* **{**
* //将name的所有first集所对应的位置填入当前推导式的编号
* mtable**.**mtable**[**mtable**.**vnname**[**vnname**]][**mtable**.**vtname**[\***iter**]]** **=** gmaper**->**first**;**
* **}**
* **}**
* //如果name可以产生空字 则将vnname所有的follow集所对应的位置填上 vnname推导出空的编号
* **if** **(**emptyflag**)**
* **{**
* **for** **(**iter **=** followlist**[**vnname**].**begin**();**iter **!=** followlist**[**vnname**].**end**();**iter**++)**
* mtable**.**mtable**[**mtable**.**vnname**[**vnname**]][**mtable**.**vtname**[\***iter**]]** **=** emptymake**[**vnname**];**
* **}**
* **}**
* //格式化输出预测分析表至文件
* out **<<** setw**(**20**)** **<<** ""**;**
* **for** **(**iter **=** vt**.**begin**();**iter **!=** vt**.**end**();**iter**++)**
* **if** **(\***iter **!=** "empty"**)**
* out **<<** setw**(**8**)** **<<** left **<<** **\***iter**;**
* out **<<** endl**;**
* **for** **(**iter **=** vn**.**begin**(),** i **=** 1**;**iter **!=** vn**.**end**();**iter**++,** i**++)**
* **{**
* out **<<** setw**(**20**)** **<<** left **<<** **\***iter**;**
* **for** **(**j **=** 1**;**j **<=** mtable**.**vtnum**;**j**++)**
* out **<<** setw**(**8**)** **<<** left **<<** mtable**.**mtable**[**i**][**j**];**
* out **<<** endl**;**
* **}**
* **}**
* //初始化栈
* void stack\_init**(**struct Stack **&**stack**)**
* **{**
* stack**.**st**.**push\_back**(**"#"**);**
* stack**.**st**.**push\_back**(**firststr**);**
* stack**.**current **=** stack**.**st**.**size**();**
* **}**
* //元素入栈
* void stack\_pushin**(**struct Stack **&**stack**,** string str**)**
* **{**
* stack**.**st**.**push\_back**(**str**);**
* stack**.**current**++;**
* **}**
* //获取栈顶元素
* string stack\_gettop**(**struct Stack stack**)**
* **{**
* **return** stack**.**st**[**stack**.**current **-** 1**];**
* **}**
* //栈顶元素出栈
* void stack\_popout**(**struct Stack **&**stack**)**
* **{**
* string str**;**
* **if** **(**stack**.**st**.**size**()** **==** 0**)**
* **{**
* cout **<<** "ERROR:栈已为空,无法出栈!" **<<** endl**;**
* exit**(**0**);**
* **}**
* **else**
* **{**
* stack**.**st**.**pop\_back**();**
* stack**.**current**--;**
* **}**
* **}**
* //输出栈中信息
* void stack\_show**(**struct Stack stack**,** fstream **&**out**)**
* **{**
* int i**;**
* out **<<** "当前栈中元素个数:" **<<** stack**.**current**;**
* **for** **(**i **=** 0**;**i **<** stack**.**current**;**i**++)**
* out **<<** " " **<<** stack**.**st**[**i**];**
* out **<<** endl**;**
* **}**
* //初始化语法树
* void tree\_init**()**
* **{**
* treenode **=** **(**treebit **\*)**malloc**(sizeof(**treebit**));**
* strcpy\_s**(**treenode**->**re**,** "head"**);**
* strcpy\_s**(**treenode**->**type**,** "derivation"**);**
* treenode**->**bitnum **=** 3**;**
* treenode**->**father **=** **NULL;**
* treenode**->**brother **=** **NULL;** //初始化根节点
* **for** **(**int i **=** 1**;**i **<=** treenode**->**bitnum**;**i**++)**
* **{**
* treenode**->**sons**[**i**]** **=** **(**struct Gtree **\*)**malloc**(sizeof(**struct Gtree**));**
* treenode**->**sons**[**i**]->**father **=** **(**struct Gtree **\*)**malloc**(sizeof(**struct Gtree**));**
* treenode**->**sons**[**i**]->**brother **=** **(**struct Gtree **\*)**malloc**(sizeof(**struct Gtree**));**
* **}**
* strcpy\_s**(**treenode**->**sons**[**1**]->**re**,** "#"**);**
* strcpy\_s**(**treenode**->**sons**[**1**]->**type**,** "#"**);**
* treenode**->**sons**[**1**]->**brother **=** treenode**->**sons**[**2**];**
* treenode**->**sons**[**1**]->**father **=** treenode**;**
* nodenum**++;**
* treenode**->**sons**[**2**]->**nodenum **=** nodenum**;**
* placelistmap**.**insert**({** nodenum**,{}** **});**
* strcpy\_s**(**treenode**->**sons**[**2**]->**re**,** firststr**.**c\_str**());**
* strcpy\_s**(**treenode**->**sons**[**2**]->**type**,** "derivation"**);**
* treenode**->**sons**[**2**]->**brother **=** treenode**->**sons**[**3**];**
* treenode**->**sons**[**2**]->**father **=** treenode**;**
* strcpy\_s**(**treenode**->**sons**[**3**]->**re**,** "#"**);**
* strcpy\_s**(**treenode**->**sons**[**3**]->**type**,** "#"**);**
* treenode**->**sons**[**3**]->**brother **=** **NULL;**
* treenode**->**sons**[**3**]->**father **=** treenode**;** //初始化根节点的孩子节点 使用推导式 firststr-># firststr #
* **}**
* //生成语法分析树
* void make\_tree**(**fstream **&**in**,** fstream **&**out**)**
* **{**
* string str**;**
* int num**,** line**;**
* struct Gtree **\***cur**;**
* tree\_init**();**
* stack\_init**(**stack**);** //初始化语法分析树以及语法栈
* cur **=** treenode**->**sons**[**2**];**
* **while** **(!**in**.**eof**())**
* **{**
* **if** **(!**stack**.**current**)**
* **break;**
* in **>>** skipws**;**
* in **>>** token**.**re **>>** token**.**type **>>** token**.**line **>>** token**.**colume**;**
* **if** **(**in**.**fail**())**
* **break;** //从词法分析结果中逐个读取token
* stack\_show**(**stack**,**out**);**
* str **=** stack\_gettop**(**stack**);**
* **while** **(**mtable**.**vtname**.**find**(**str**)** **==** mtable**.**vtname**.**end**())**
* **{** //直到栈顶元素为终结符 跳出循环 准备比较\出栈操作
* **if** **(**token**.**type **!=** "NUM" **&&** token**.**type **!=** "ID" **&&** token**.**type **!=** "IDF" **&&** token**.**type **!=** "ID1"**)**
* num **=** mtable**.**vtname**[**token**.**re**];**
* **else**
* num **=** mtable**.**vtname**[**token**.**type**];** //数字 ID类型token通过原始字符串得到编号 其余通过type类型
* **if** **(**mtable**.**mtable**[**mtable**.**vnname**[**str**]][**num**]** **==** 0**)**
* **{**
* out **<<** "Grammar Error at Line " **<<** token**.**line **<<** " Colume " **<<** token**.**colume **<<** endl**;**
* cout **<<** "Grammar Error at Line " **<<** token**.**line **<<** " Colume " **<<** token**.**colume **<<** endl**;**
* //cout << str << endl;
* //cout << token.re << endl;
* exit**(**0**);**
* **}** //预测分析表行列所对应的推导式标号为0 表示出现语法错误
* **else**
* **{**
* line **=** mtable**.**mtable**[**mtable**.**vnname**[**str**]][**num**];**
* stack\_popout**(**stack**);**
* cur**->**bitnum **=** 0**;**
* cur**->**mtnum **=** line**;**
* **for** **(**iter **=** grammarmap**[**line**].**end**()** **-** 1**;**iter **!=** grammarmap**[**line**].**begin**();**iter**--)**
* **{**
* **if** **(\***iter **!=** "empty"**)**
* **{**
* stack\_pushin**(**stack**,** **\***iter**);**
* **}**
* **}** //将不为empty的产生式 压入栈中
* **for** **(**iter **=** grammarmap**[**line**].**begin**()** **+** 1**;**iter **!=** grammarmap**[**line**].**end**();**iter**++)**
* **{**
* **if** **(**cur**->**mtnum **==** 42**)**
* **{**
* **if** **(\***iter **==** "expression"**)**
* **{**
* cur**->**bitnum**++;**
* cur**->**sons**[**cur**->**bitnum**]** **=** **(**treebit **\*)**malloc**(sizeof(**treebit**));**
* cur**->**sons**[**cur**->**bitnum **-** 1**]->**brother **=** cur**->**sons**[**cur**->**bitnum**];**
* cur**->**sons**[**cur**->**bitnum**]->**father **=** cur**;**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**re**,** "$42\_1"**);**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**type**,** "todo"**);**
* **}**
* **if** **(\***iter **==** ")"**)**
* **{**
* cur**->**bitnum**++;**
* cur**->**sons**[**cur**->**bitnum**]** **=** **(**treebit **\*)**malloc**(sizeof(**treebit**));**
* cur**->**sons**[**cur**->**bitnum **-** 1**]->**brother **=** cur**->**sons**[**cur**->**bitnum**];**
* cur**->**sons**[**cur**->**bitnum**]->**father **=** cur**;**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**re**,** "$42\_2"**);**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**type**,** "todo"**);**
* **}**
* //42 iteration-stmt -> while ( $42\_1 expression $42\_2 ) statement $42\_3
* **}** //while语句推导中需要增加两个中间语义动作
* **if** **(**cur**->**mtnum **==** 39**)**
* **{**
* **if** **(\***iter **==** ")"**)**
* **{**
* cur**->**bitnum**++;**
* cur**->**sons**[**cur**->**bitnum**]** **=** **(**treebit **\*)**malloc**(sizeof(**treebit**));**
* cur**->**sons**[**cur**->**bitnum **-** 1**]->**brother **=** cur**->**sons**[**cur**->**bitnum**];**
* cur**->**sons**[**cur**->**bitnum**]->**father **=** cur**;**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**re**,** "$39\_1"**);**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**type**,** "todo"**);**
* **}**
* **if** **(\***iter **==** "selection-stmt'"**)**
* **{**
* cur**->**bitnum**++;**
* cur**->**sons**[**cur**->**bitnum**]** **=** **(**treebit **\*)**malloc**(sizeof(**treebit**));**
* cur**->**sons**[**cur**->**bitnum **-** 1**]->**brother **=** cur**->**sons**[**cur**->**bitnum**];**
* cur**->**sons**[**cur**->**bitnum**]->**father **=** cur**;**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**re**,** "$39\_2"**);**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**type**,** "todo"**);**
* **}** //39 selection-stmt -> if ( expression $39\_1 ) statement $39\_2 selection-stmt' $39\_3
* **}** //if语句推导中需要增加两个中间语义动作
* **if** **(**cur**->**mtnum **==** 41**)**
* **{**
* **if** **(\***iter **==** "statement"**)**
* **{**
* cur**->**bitnum**++;**
* cur**->**sons**[**cur**->**bitnum**]** **=** **(**treebit **\*)**malloc**(sizeof(**treebit**));**
* cur**->**sons**[**cur**->**bitnum **-** 1**]->**brother **=** cur**->**sons**[**cur**->**bitnum**];**
* cur**->**sons**[**cur**->**bitnum**]->**father **=** cur**;**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**re**,** "$41\_1"**);**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**type**,** "todo"**);**
* **}** //41 selection-stmt' -> else $41\_1 statement $41\_2
* **}** //else语句推导中需要增加一个中间语义动作
* cur**->**bitnum**++;**
* cur**->**sons**[**cur**->**bitnum**]** **=** **(**treebit **\*)**malloc**(sizeof(**treebit**));**
* **if** **(**cur**->**bitnum **>** 1**)**
* cur**->**sons**[**cur**->**bitnum **-** 1**]->**brother **=** cur**->**sons**[**cur**->**bitnum**];**
* cur**->**sons**[**cur**->**bitnum**]->**father **=** cur**;**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**re**,** **(\***iter**).**c\_str**());**
* **if** **(**mtable**.**vnname**.**find**(\***iter**)** **!=** mtable**.**vnname**.**end**())**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**type**,** "derivation"**);**
* **else**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**type**,** vttype**.**find**(\***iter**)->**second**.**c\_str**());**
* cur**->**sons**[**cur**->**bitnum**]->**info **=** **(**struct info **\*)**malloc**(sizeof(**struct info**));**
* nodenum**++;**
* cur**->**sons**[**cur**->**bitnum**]->**nodenum **=** nodenum**;**
* placelistmap**.**insert**({** nodenum**,{}** **});**
* **}** //将该条产生式的所有元素 加入到当前节点的孩子节点 并填充相关信息以及兄弟父亲节点
* cur**->**bitnum**++;**
* stringstream ss**;**
* ss **<<** cur**->**mtnum**;**
* cur**->**sons**[**cur**->**bitnum**]** **=** **(**treebit **\*)**malloc**(sizeof(**treebit**));**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**re**,** **(**"$" **+** ss**.**str**()).**c\_str**());**
* **if** **(**ss**.**str**()** **==** "42"**)**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**re**,** "$42\_3"**);**
* **if** **(**ss**.**str**()** **==** "39"**)**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**re**,** "$39\_3"**);**
* **if** **(**ss**.**str**()** **==** "41"**)**
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**re**,** "$41\_2"**);**
* //语义分析节点命名由 $ 符号 加上 产生式编号组成
* strcpy\_s**(**cur**->**sons**[**cur**->**bitnum**]->**type**,** "todo"**);**
* cur**->**sons**[**cur**->**bitnum**]->**father **=** cur**;**
* cur**->**sons**[**cur**->**bitnum **-** 1**]->**brother **=** cur**->**sons**[**cur**->**bitnum**];**
* cur**->**sons**[**cur**->**bitnum**]->**brother **=** **NULL;** //最后添加$节点 表明用第几条产生式产生 便于后续语义分析
* cur **=** cur**->**sons**[**1**];**
* **if** **(**strcmp**(**cur**->**re**,** "empty"**)** **==** 0**)**
* **{**
* **while** **(**1**)**
* **{**
* **while** **(!**cur**->**brother**)**
* **{**
* cur **=** cur**->**father**;**
* **if** **(**strcmp**(**cur**->**re**,** "head"**)** **==** 0**)**
* **break;**
* **}**
* cur **=** cur**->**brother**;**
* **if** **(**cur**->**re**[**0**]** **!=** '$'**)**
* **break;**
* **}**
* **}** //应用完当前产生式 进入兄弟节点继续推导 没有兄弟节点 返回父亲节点 进入它的兄弟节点 知道兄弟节点存在
* str **=** stack\_gettop**(**stack**);**
* stack\_show**(**stack**,** out**);**
* **}**
* **}**
* **if** **(**token**.**re **!=** stack\_gettop**(**stack**)** **&&** token**.**type **!=** stack\_gettop**(**stack**))**
* **{**
* out **<<** "Grammar Error at Line " **<<** token**.**line **<<** " Colume " **<<** token**.**colume **<<** endl**;**
* cout **<<** "Grammar Error at Line " **<<** token**.**line **<<** " Colume " **<<** token**.**colume **<<** endl**;**
* exit**(**0**);**
* **}** //栈顶终结符类型与当前token类型不符合 报错
* **if** **(**strstr**(**cur**->**type**,** "ID"**))**
* strcpy\_s**(**cur**->**re**,** token**.**re**.**c\_str**());** //对于id类型直接改变当前节点原始字符串 不再产生新的节点
* **if** **(**strstr**(**cur**->**type**,** "NUM"**))**
* strcpy\_s**(**cur**->**re**,** token**.**re**.**c\_str**());**
* **while** **(**1**)**
* **{**
* **while** **(!**cur**->**brother**)**
* **{**
* cur **=** cur**->**father**;**
* **if** **(**strcmp**(**cur**->**re**,** "head"**)** **==** 0**)**
* **break;**
* **}**
* **if** **(**strcmp**(**cur**->**re**,** "head"**)** **==** 0**)**
* **break;**
* cur **=** cur**->**brother**;**
* **if** **(**cur**->**re**[**0**]** **!=** '$'**)**
* **break;**
* **}** //寻找满足条件的兄弟节点
* //栈顶元素出栈
* stack\_popout**(**stack**);**
* **}**
* **}**
* //产生一个新的临时变量
* string newtemp**()**
* **{**
* string T **=** "T"**;**
* stringstream ss**;**
* ss **<<** curtemp**;**
* curtemp**++;** //临时变量个数加一
* T**.**append**(**ss**.**str**());** //临时变量命名规则 T+当前个数
* **return** T**;**
* **}**
* //输出中间代码
* void printmidcode**(**fstream **&**out**)**
* **{**
* map**<**int**,** string**>::**iterator iter\_i\_s**;**
* **for** **(**iter\_i\_s **=** midcodemap**.**begin**();**iter\_i\_s **!=** midcodemap**.**end**();**iter\_i\_s**++)**
* out **<<** "(" **<<** iter\_i\_s**->**first **<<** ") " **<<** iter\_i\_s**->**second **<<** endl**;**
* **}**
* bool fomula\_1**(**treebit **\***temp**)**
* **{**
* //1 S -> program
* **return** **true;**
* **}**
* bool fomula\_2**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_3**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_4**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_5**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_6**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_7**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_8**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_9**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_10**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_11**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_12**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_13**(**treebit **\***temp**)**
* **{**
* //13 fun-declaration -> type-specifier IDF ( params )
* temp **=** temp**->**father**;**
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**proc **+** temp**->**sons**[**2**]->**re **});** //插入中间代码 定义函数
* **return** **true;**
* **}**
* bool fomula\_14**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_15**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_16**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_17**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_18**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_19**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_20**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_21**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_22**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_23**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_24**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_25**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_26**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_27**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_28**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_29**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_30**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_31**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_32**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_33**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_34**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_35**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_36**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_37**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_38**(**treebit **\***temp**)**
* **{**
* **return** **true;**
* **}**
* bool fomula\_39\_1**(**treebit **\***temp**)**
* **{**
* //39 selection-stmt -> if ( expression $39\_1 ) statement $39\_2 selection-stmt' $39\_3
* temp **=** temp**->**father**;**
* temp**->**info**->**ifnum1 **=** curnum **+** 2**;** //if语句真出口为当前curnum+2的位置
* curnum**++;**
* stringstream ss**;**
* ss **<<** curnum **+** 2**;**
* midcodemap**.**insert**({** curnum**,**"if " **+** placelistmap**[**temp**->**sons**[**3**]->**nodenum**][**0**]** **+** "==1 Goto " **+** ss**.**str**()** **});**
* //为真是强制跳转当前位置+2的位置
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**"Goto " **});**
* //为假强制跳转至假出口 暂时没有求出
* **return** **true;**
* **}**
* bool fomula\_39\_2**(**treebit **\***temp**)**
* **{**
* //39 selection-stmt -> if ( expression $39\_1 ) statement $39\_2 selection-stmt' $39\_3
* temp **=** temp**->**father**;**
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**"Goto " **});** //if语句为真执行语句结束 强制跳转至if语句下一条 暂时未求出
* temp**->**info**->**ifnum2 **=** curnum**;** //得到if假出口位置前一条
* **return** **true;**
* **}**
* bool fomula\_39\_3**(**treebit **\***temp**)**
* **{**
* //39 selection-stmt -> if ( expression $39\_1 ) statement $39\_2 selection-stmt' $39\_3
* temp **=** temp**->**father**;**
* **if** **(**temp**->**sons**[**8**]->**info**->**emptyflag**)**
* **{**
* stringstream ss**;**
* ss **<<** temp**->**info**->**ifnum2**;**
* midcodemap**[**temp**->**info**->**ifnum1**].**append**(**ss**.**str**());**
* midcodemap**.**erase**(**curnum**);**
* curnum**--;**
* **}** //emptyflag为1即不存在else语句 删除最后一天强制跳转语句 并且将之前空着的判断条件为假时的跳转位置填上
* **else**
* **{**
* stringstream ss**;**
* ss **<<** temp**->**sons**[**8**]->**info**->**elsenum1 **+** 1**;**
* midcodemap**[**temp**->**info**->**ifnum1**].**append**(**ss**.**str**());**
* stringstream s**;**
* s **<<** temp**->**sons**[**8**]->**info**->**elsenum2 **+** 1**;**
* midcodemap**[**temp**->**info**->**ifnum2**].**append**(**s**.**str**());**
* **}** //emptyflag为0存在else语句 填充判断条件为假的跳转地址以及为真代码块结束的强制跳转地址
* **return** **true;**
* **}**
* bool fomula\_40**(**treebit **\***temp**)**
* **{**
* //40 selection-stmt' -> empty
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 1**;** //无else语句将emptyflag赋为0
* **return** **true;**
* **}**
* bool fomula\_41\_1**(**treebit **\***temp**)**
* **{**
* //41 selection-stmt' -> else $41\_1 statement $41\_2
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 0**;**
* temp**->**info**->**elsenum1 **=** curnum**;** //记录else的入口地址 在43\_3时填入
* **return** **true;**
* **}**
* bool fomula\_41\_2**(**treebit **\***temp**)**
* **{**
* //41 selection-stmt' -> else $41\_1 statement $41\_2
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 0**;**
* temp**->**info**->**elsenum2 **=** curnum**;** //记录else的出口地址 在43\_3时填入
* **return** **true;**
* **}**
* bool fomula\_42\_1**(**treebit **\***temp**)**
* **{**
* //42 iteration-stmt -> while ( $42\_1 expression $42\_2 ) statement $42\_3
* temp **=** temp**->**father**;**
* temp**->**info**->**whilenum1 **=** curnum **+** 1**;** //记录while入口地址 42\_3时跳转回此地址
* **return** **true;**
* **}**
* bool fomula\_42\_2**(**treebit **\***temp**)**
* **{**
* //42 iteration-stmt -> while ( $42\_1 expression $42\_2 ) statement $42\_3
* temp **=** temp**->**father**;**
* temp**->**info**->**whilenum2 **=** curnum **+** 1**;**
* curnum**++;**
* stringstream ss**;**
* ss **<<** curnum **+** 2**;**
* midcodemap**.**insert**({** curnum**,**"if " **+** placelistmap**[**temp**->**sons**[**4**]->**nodenum**][**0**]** **+** "==1 Goto " **+** ss**.**str**()** **});**
* //插入中间代码判断expression是否为真 为真强制跳转至当前地址+2
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**"Goto " **});**
* //为假跳转至假出口 暂时为求出
* **return** **true;**
* **}**
* bool fomula\_42\_3**(**treebit **\***temp**)**
* **{**
* //42 iteration-stmt -> while ( $42\_1 expression $42\_2 ) statement $42\_3
* temp **=** temp**->**father**;**
* stringstream ss**;**
* ss **<<** curnum **+** 2**;**
* midcodemap**[**temp**->**info**->**whilenum2 **+** 1**].**append**(**ss**.**str**());**
* //回填假出口值
* curnum**++;**
* stringstream s**;**
* s **<<** temp**->**info**->**whilenum1**;**
* midcodemap**.**insert**({** curnum**,**"Goto " **+** s**.**str**()** **});**
* //插入中间代码 强制跳转回42\_1位置
* **return** **true;**
* **}**
* bool fomula\_43**(**treebit **\***temp**)**
* **{**
* //43 return-stmt -> return return-stmt'
* temp **=** temp**->**father**;**
* temp**->**info**->**returnflag **=** 1**;**
* **if** **(**temp**->**sons**[**2**]->**info**->**emptyflag**)**
* **{**
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**"return" **});**
* **}** //无返回值
* **else**
* **{**
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**"return " **+** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]** **});**
* **}** //有返回值
* **return** **true;**
* **}**
* bool fomula\_44**(**treebit **\***temp**)**
* **{**
* //44 return-stmt' -> ;
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 1**;**
* **return** **true;**
* **}**
* bool fomula\_45**(**treebit **\***temp**)**
* **{**
* //45 return-stmt' -> expression ;
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 0**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]);**
* **return** **true;**
* **}**
* bool fomula\_46**(**treebit **\***temp**)**
* **{**
* //46 expression -> var1 = expression
* temp **=** temp**->**father**;**
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]** **+** " := " **+** placelistmap**[**temp**->**sons**[**3**]->**nodenum**][**0**]** **});**
* **return** **true;**
* **}**
* bool fomula\_47**(**treebit **\***temp**)**
* **{**
* //47 expression -> simple-expression
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]);**
* **return** **true;**
* **}**
* bool fomula\_48**(**treebit **\***temp**)**
* **{**
* //48 var -> ID var'
* temp **=** temp**->**father**;**
* char ts**[**255**];**
* **if** **(**temp**->**sons**[**2**]->**info**->**arrayflag**)**
* **{**
* strcat\_s**(**ts**,** temp**->**sons**[**1**]->**re**);**
* strcat\_s**(**ts**,** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**].**c\_str**());**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**ts**);**
* **}**
* **else**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**temp**->**sons**[**1**]->**re**);**
* **return** **true;**
* **}**
* bool fomula\_49**(**treebit **\***temp**)**
* **{**
* //49 var' -> empty
* temp **=** temp**->**father**;**
* temp**->**info**->**arrayflag **=** 0**;**
* **return** **true;**
* **}**
* bool fomula\_50**(**treebit **\***temp**)**
* **{**
* //50 var' -> [ expression ]
* temp **=** temp**->**father**;**
* temp**->**info**->**arrayflag **=** 1**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**"[" **+** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]** **+** "]"**);**
* //temp->info.placelist.push\_back("[" + temp->sons[2]->info.placelist[0] + "]");
* **return** **true;**
* **}**
* bool fomula\_51**(**treebit **\***temp**)**
* **{**
* //51 var1 -> ID1 var1'
* temp **=** temp**->**father**;**
* char ts**[**255**];**
* **if** **(**temp**->**sons**[**2**]->**info**->**arrayflag**)**
* **{**
* strcat\_s**(**ts**,** temp**->**sons**[**1**]->**re**);**
* strcat\_s**(**ts**,** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**].**c\_str**());**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**ts**);**
* **}**
* **else**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**temp**->**sons**[**1**]->**re**);**
* **return** **true;**
* **}**
* bool fomula\_52**(**treebit **\***temp**)**
* **{**
* //52 var1' -> empty
* temp **=** temp**->**father**;**
* temp**->**info**->**arrayflag **=** 0**;**
* **return** **true;**
* **}**
* bool fomula\_53**(**treebit **\***temp**)**
* **{**
* //53 var1' -> [ expression ]
* temp **=** temp**->**father**;**
* temp**->**info**->**arrayflag **=** 1**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**"[" **+** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]** **+** "]"**);**
* **return** **true;**
* **}**
* bool fomula\_54**(**treebit **\***temp**)**
* **{**
* //54 simple-expression -> additive-expression simple-expression'
* temp **=** temp**->**father**;**
* **if** **(**temp**->**sons**[**2**]->**info**->**emptyflag**)**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]);**
* **else**
* **{**
* string t **=** newtemp**();**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**t**);**
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**t **+** " := " **+** placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]** **+** " " **+** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]** **});**
* **}**
* **return** **true;**
* **}**
* bool fomula\_55**(**treebit **\***temp**)**
* **{**
* //55 simple-expression' -> relop additive-expression
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 0**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]** **+** " " **+** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]);**
* **return** **true;**
* **}**
* bool fomula\_56**(**treebit **\***temp**)**
* **{**
* //56 simple-expression' -> empty
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 1**;**
* **return** **true;**
* **}**
* bool fomula\_57**(**treebit **\***temp**)**
* **{**
* //relop <=
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**"<="**);**
* **return** **true;**
* **}**
* bool fomula\_58**(**treebit **\***temp**)**
* **{**
* //relop <
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**"<"**);**
* **return** **true;**
* **}**
* bool fomula\_59**(**treebit **\***temp**)**
* **{**
* //relop >
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**">"**);**
* **return** **true;**
* **}**
* bool fomula\_60**(**treebit **\***temp**)**
* **{**
* //relop >=
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**">="**);**
* **return** **true;**
* **}**
* bool fomula\_61**(**treebit **\***temp**)**
* **{**
* //rolop ==
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**"=="**);**
* **return** **true;**
* **}**
* bool fomula\_62**(**treebit **\***temp**)**
* **{**
* //relop !=
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**"!="**);**
* **return** **true;**
* **}**
* bool fomula\_63**(**treebit **\***temp**)**
* **{**
* //63 additive-expression -> term additive-expression'
* temp **=** temp**->**father**;**
* **if** **(**temp**->**sons**[**2**]->**info**->**emptyflag**)**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]);**
* **else**
* **{**
* string t **=** newtemp**();**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**t**);**
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**t **+** " := " **+** placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]** **+** " " **+** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]** **});**
* **}**
* **return** **true;**
* **}**
* bool fomula\_64**(**treebit **\***temp**)**
* **{**
* //64 additive-expression' -> addop term additive-expression'
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 0**;**
* **if** **(**temp**->**sons**[**3**]->**info**->**emptyflag**)**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]** **+** " " **+** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]);**
* **else**
* **{**
* string t **=** newtemp**();**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]** **+** " " **+** t**);**
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**t **+** " := " **+** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]** **+** " " **+** placelistmap**[**temp**->**sons**[**3**]->**nodenum**][**0**]** **});**
* **}**
* **return** **true;**
* **}**
* bool fomula\_65**(**treebit **\***temp**)**
* **{**
* //65 additive-expression' -> empty
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 1**;**
* **return** **true;**
* **}**
* bool fomula\_66**(**treebit **\***temp**)**
* **{**
* //addop +
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**"+"**);**
* **return** **true;**
* **}**
* bool fomula\_67**(**treebit **\***temp**)**
* **{**
* //addop -
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**"-"**);**
* **return** **true;**
* **}**
* bool fomula\_68**(**treebit **\***temp**)**
* **{**
* //68 term -> factor term'
* temp **=** temp**->**father**;**
* **if** **(**temp**->**sons**[**2**]->**info**->**emptyflag**)**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]);**
* **else**
* **{**
* string t **=** newtemp**();**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**t**);**
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**t **+** " := " **+** placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]** **+** " " **+** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]** **});**
* **}**
* **return** **true;**
* **}**
* bool fomula\_69**(**treebit **\***temp**)**
* **{**
* //69 term' -> mulop factor term'
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 0**;**
* **if** **(**temp**->**sons**[**3**]->**info**->**emptyflag**)**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]** **+** " " **+** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]);**
* **else**
* **{**
* string t**=**newtemp**();**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]** **+** " " **+** t**);**
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**t **+** " := " **+** placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]** **+** " " **+** placelistmap**[**temp**->**sons**[**3**]->**nodenum**][**0**]** **});**
* **}**
* **return** **true;**
* **}**
* bool fomula\_70**(**treebit **\***temp**)**
* **{**
* //70 term' -> empty
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 1**;**
* **return** **true;**
* **}**
* bool fomula\_71**(**treebit **\***temp**)**
* **{**
* //mulop \*
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**"\*"**);**
* **return** **true;**
* **}**
* bool fomula\_72**(**treebit **\***temp**)**
* **{**
* //mulop /
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**"/"**);**
* **return** **true;**
* **}**
* bool fomula\_73**(**treebit **\***temp**)**
* **{**
* //73 factor -> ( expression )
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]);**
* **return** **true;**
* **}**
* bool fomula\_74**(**treebit **\***temp**)**
* **{**
* //74 factor -> var
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]);**
* **return** **true;**
* **}**
* bool fomula\_75**(**treebit **\***temp**)**
* **{**
* //75 factor -> call
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]);**
* **return** **true;**
* **}**
* bool fomula\_76**(**treebit **\***temp**)**
* **{**
* //76 factor -> NUM
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**temp**->**sons**[**1**]->**re**);**
* **return** **true;**
* **}**
* bool fomula\_77**(**treebit **\***temp**)**
* **{**
* //77 call -> IDF ( args )
* temp **=** temp**->**father**;**
* string t**;**
* t **=** newtemp**();**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**t**);**
* **if** **(!**temp**->**sons**[**3**]->**info**->**emptyflag**)**
* **for** **(**iter **=** placelistmap**[**temp**->**sons**[**3**]->**nodenum**].**begin**();**iter **!=** placelistmap**[**temp**->**sons**[**3**]->**nodenum**].**end**();**iter**++)**
* **{**
* curnum**++;**
* midcodemap**.**insert**({** curnum**,**param **+** **\***iter **});**
* **}**
* curnum**++;**
* midcodemap**.**insert**({** curnum **,**t **+** " := " **+** call **+** temp**->**sons**[**1**]->**re **});**
* **return** **true;**
* **}**
* bool fomula\_78**(**treebit **\***temp**)**
* **{**
* //78 args -> arg-list
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 0**;**
* **for** **(**iter **=** placelistmap**[**temp**->**sons**[**1**]->**nodenum**].**begin**();**iter **!=** placelistmap**[**temp**->**sons**[**1**]->**nodenum**].**end**();**iter**++)**
* placelistmap**[**temp**->**nodenum**].**push\_back**(\***iter**);**
* **return** **true;**
* **}**
* bool fomula\_79**(**treebit **\***temp**)**
* **{**
* //79 args -> empty
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 1**;**
* **return** **true;**
* **}**
* bool fomula\_80**(**treebit **\***temp**)**
* **{**
* //80 arg-list -> expression arg-list'
* temp **=** temp**->**father**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**1**]->**nodenum**][**0**]);**
* **if** **(!**temp**->**sons**[**2**]->**info**->**emptyflag**)**
* **for** **(**iter **=** placelistmap**[**temp**->**sons**[**2**]->**nodenum**].**begin**();**iter **!=** placelistmap**[**temp**->**sons**[**2**]->**nodenum**].**end**();**iter**++)**
* placelistmap**[**temp**->**nodenum**].**push\_back**(\***iter**);**
* **return** **true;**
* **}**
* bool fomula\_81**(**treebit **\***temp**)**
* **{**
* //81 arg-list' -> , expression arg-list'
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 0**;**
* placelistmap**[**temp**->**nodenum**].**push\_back**(**placelistmap**[**temp**->**sons**[**2**]->**nodenum**][**0**]);**
* **if** **(!**temp**->**sons**[**3**]->**info**->**emptyflag**)**
* **for** **(**iter **=** placelistmap**[**temp**->**sons**[**3**]->**nodenum**].**begin**();**iter **!=** placelistmap**[**temp**->**sons**[**3**]->**nodenum**].**end**();**iter**++)**
* placelistmap**[**temp**->**nodenum**].**push\_back**(\***iter**);**
* **return** **true;**
* **}**
* bool fomula\_82**(**treebit **\***temp**)**
* **{**
* //82 arg-list' -> empty
* temp **=** temp**->**father**;**
* temp**->**info**->**emptyflag **=** 1**;**
* **return** **true;**
* **}**
* //初始化语义动作表
* void init\_funcmap**()**
* **{**
* funcmap**.**insert**({**"$1"**,**fomula\_1**});**
* funcmap**.**insert**({** "$2"**,**fomula\_2 **});**
* funcmap**.**insert**({** "$3"**,**fomula\_3 **});**
* funcmap**.**insert**({** "$4"**,**fomula\_4 **});**
* funcmap**.**insert**({** "$5"**,**fomula\_5 **});**
* funcmap**.**insert**({** "$6"**,**fomula\_6 **});**
* funcmap**.**insert**({** "$7"**,**fomula\_7 **});**
* funcmap**.**insert**({** "$8"**,**fomula\_8 **});**
* funcmap**.**insert**({** "$9"**,**fomula\_9 **});**
* funcmap**.**insert**({** "$10"**,**fomula\_10 **});**
* funcmap**.**insert**({** "$11"**,**fomula\_11 **});**
* funcmap**.**insert**({** "$12"**,**fomula\_12 **});**
* funcmap**.**insert**({** "$13"**,**fomula\_13 **});**
* funcmap**.**insert**({** "$14"**,**fomula\_14 **});**
* funcmap**.**insert**({** "$15"**,**fomula\_15 **});**
* funcmap**.**insert**({** "$16"**,**fomula\_16 **});**
* funcmap**.**insert**({** "$17"**,**fomula\_17 **});**
* funcmap**.**insert**({** "$18"**,**fomula\_18 **});**
* funcmap**.**insert**({** "$19"**,**fomula\_19 **});**
* funcmap**.**insert**({** "$20"**,**fomula\_20 **});**
* funcmap**.**insert**({** "$21"**,**fomula\_21 **});**
* funcmap**.**insert**({** "$22"**,**fomula\_22 **});**
* funcmap**.**insert**({** "$23"**,**fomula\_23 **});**
* funcmap**.**insert**({** "$24"**,**fomula\_24 **});**
* funcmap**.**insert**({** "$25"**,**fomula\_25 **});**
* funcmap**.**insert**({** "$26"**,**fomula\_26 **});**
* funcmap**.**insert**({** "$27"**,**fomula\_27 **});**
* funcmap**.**insert**({** "$28"**,**fomula\_28 **});**
* funcmap**.**insert**({** "$29"**,**fomula\_29 **});**
* funcmap**.**insert**({** "$30"**,**fomula\_30 **});**
* funcmap**.**insert**({** "$31"**,**fomula\_31 **});**
* funcmap**.**insert**({** "$32"**,**fomula\_32 **});**
* funcmap**.**insert**({** "$33"**,**fomula\_33 **});**
* funcmap**.**insert**({** "$34"**,**fomula\_34 **});**
* funcmap**.**insert**({** "$35"**,**fomula\_35 **});**
* funcmap**.**insert**({** "$36"**,**fomula\_36 **});**
* funcmap**.**insert**({** "$37"**,**fomula\_37 **});**
* funcmap**.**insert**({** "$38"**,**fomula\_38 **});**
* funcmap**.**insert**({** "$39\_1"**,**fomula\_39\_1 **});**
* funcmap**.**insert**({** "$39\_2"**,**fomula\_39\_2 **});**
* funcmap**.**insert**({** "$39\_3"**,**fomula\_39\_3 **});**
* funcmap**.**insert**({** "$40"**,**fomula\_40 **});**
* funcmap**.**insert**({** "$41\_1"**,**fomula\_41\_1 **});**
* funcmap**.**insert**({** "$41\_2"**,**fomula\_41\_2 **});**
* funcmap**.**insert**({** "$42\_1"**,**fomula\_42\_1 **});**
* funcmap**.**insert**({** "$42\_2"**,**fomula\_42\_2 **});**
* funcmap**.**insert**({** "$42\_3"**,**fomula\_42\_3 **});**
* funcmap**.**insert**({** "$43"**,**fomula\_43 **});**
* funcmap**.**insert**({** "$44"**,**fomula\_44 **});**
* funcmap**.**insert**({** "$45"**,**fomula\_45 **});**
* funcmap**.**insert**({** "$46"**,**fomula\_46 **});**
* funcmap**.**insert**({** "$47"**,**fomula\_47 **});**
* funcmap**.**insert**({** "$48"**,**fomula\_48 **});**
* funcmap**.**insert**({** "$49"**,**fomula\_49 **});**
* funcmap**.**insert**({** "$50"**,**fomula\_50 **});**
* funcmap**.**insert**({** "$51"**,**fomula\_51 **});**
* funcmap**.**insert**({** "$52"**,**fomula\_52 **});**
* funcmap**.**insert**({** "$53"**,**fomula\_53 **});**
* funcmap**.**insert**({** "$54"**,**fomula\_54 **});**
* funcmap**.**insert**({** "$55"**,**fomula\_55 **});**
* funcmap**.**insert**({** "$56"**,**fomula\_56 **});**
* funcmap**.**insert**({** "$57"**,**fomula\_57 **});**
* funcmap**.**insert**({** "$58"**,**fomula\_58 **});**
* funcmap**.**insert**({** "$59"**,**fomula\_59 **});**
* funcmap**.**insert**({** "$60"**,**fomula\_60 **});**
* funcmap**.**insert**({** "$61"**,**fomula\_61 **});**
* funcmap**.**insert**({** "$62"**,**fomula\_62 **});**
* funcmap**.**insert**({** "$63"**,**fomula\_63 **});**
* funcmap**.**insert**({** "$64"**,**fomula\_64 **});**
* funcmap**.**insert**({** "$65"**,**fomula\_65 **});**
* funcmap**.**insert**({** "$66"**,**fomula\_66 **});**
* funcmap**.**insert**({** "$67"**,**fomula\_67 **});**
* funcmap**.**insert**({** "$68"**,**fomula\_68 **});**
* funcmap**.**insert**({** "$69"**,**fomula\_69 **});**
* funcmap**.**insert**({** "$70"**,**fomula\_70 **});**
* funcmap**.**insert**({** "$71"**,**fomula\_71 **});**
* funcmap**.**insert**({** "$72"**,**fomula\_72 **});**
* funcmap**.**insert**({** "$73"**,**fomula\_73 **});**
* funcmap**.**insert**({** "$74"**,**fomula\_74 **});**
* funcmap**.**insert**({** "$75"**,**fomula\_75 **});**
* funcmap**.**insert**({** "$76"**,**fomula\_76 **});**
* funcmap**.**insert**({** "$77"**,**fomula\_77 **});**
* funcmap**.**insert**({** "$78"**,**fomula\_78 **});**
* funcmap**.**insert**({** "$79"**,**fomula\_79 **});**
* funcmap**.**insert**({** "$80"**,**fomula\_80 **});**
* funcmap**.**insert**({** "$81"**,**fomula\_81 **});**
* funcmap**.**insert**({** "$82"**,**fomula\_82 **});**
* **}**
* //使用深度优先遍历访问语法分析树
* void deepsearch**(**treebit **\***temp**,**fstream **&**out**)**
* **{**
* **if** **(**strstr**(**temp**->**type**,** "derivation"**)** **!=** 0**)**
* **{**
* //out << temp->re << " " << temp->nodenum << endl;
* **for** **(**int i **=** 1**;**i **<=** temp**->**bitnum**;**i**++)**
* deepsearch**(**temp**->**sons**[**i**],**out**);**
* **}**
* **else**
* **{**
* //out << temp->re << endl;
* stringstream ss**;**
* ss **<<** temp**->**re**;**
* **if** **(**ss**.**str**().**find**(**"$"**)** **!=** ss**.**str**().**npos**)**
* funcmap**[**ss**.**str**()](**temp**);**
* **}**
* **}**
* //生成中间代码
* void midcode**(**fstream **&**in**,** fstream **&**out**)**
* **{**
* treebit **\***temp**;**
* init\_funcmap**();**
* temp **=** treenode**;**
* deepsearch**(**temp**->**sons**[**2**],**out**);**
* printmidcode**(**out**);**
* **}**