成绩_____

南京信息工程大学

2019-2020 第 2 学期

《编译原理》实验二分组报告

题　　目：　基于 EBNF 的编译程序设计、优化与实现

学　　院：　计算机与软件学院

专　　业：　软件工程

年级班级：　17 级 3 班、17 级 3 班

课题成员　梅承　陈飞

任课教师：　郑关胜

二〇二〇 年 五 月 二十五 日

实践小组成员分工及职责	
学号姓名	任务分工
20171344137 梅承	设计、化简语法规则，实现语法分析，报告撰写。
20171344106 陈飞	实现词法分析，绘制图标，报告撰写。

目 录

基于 EBNF 的编译程序设计、优化与实现

梅承、陈飞

南京信息工程大学计算机与软件工程，南京 210044

摘要：本文首先使用了扩展巴克斯范式（EBNF）定义了词法和语法规则，然后将这些规则通过替换、化简、消除左递归、提取公共左因子、消除冗余、消除多余产生式等手段将其转换为 LL（1）文法，当然并不是所有的文法都可以转换为 LL（1）文法，所以我们对语法规则进行了限制，使其可以顺利的转换为 LL（1）文法。然后对源程序进行词法分析，得到单词序列作为语法分析的输入。自上而下的分析的实现主要有两种，本文采用了预测分析法。的由于预测分析法的步骤为依次计算 First 集合，Follow 集合和预测分析表，然后根据预测分析表来实现语法分析，本文也对其计算过程进行了优化，采用了迭代，使其计算效率更高。

关键词：EBNF，词法分析，语法分析，预测分析过程优化。

1 引言

像用自然语言书写的文章一样，源程序是由一系列的句子组成的，句子是由单词符号按一定的规则构成的（语法规则），而单词符号又是由字符按照一定的规则构成的。因此，源程序实际上是由满足程序语言规范的字符按照一定的规则组合起来构成的一个字符串。

词法分析的功能是，从左到右逐个扫描源程序的字符串，按照词法规则，识别出单词符号（Token）作为输出，对识别过程中发现的词法错误，输出有关的错误信息。

由词法分析识别出的单词流是语法分析的输入，语法分析据此判断它们是否构成了合法的句子。

词法分析可以作为单独的一遍，这时词法分析器的输出形成一个输出文件，作为语法分析器的输入文件，词法分析也可以作为语法分析的一个子程序，每当语法分析需要下一个新单词时，就调用词法分析子程序，从输入字符串中识别一个单词后返回。在本实验中，我们将词法分析作为单独的一遍来处理。而语法分析则是使用 LL(1)分析方法。整个实验的流程如图 1 所示。

图 1　实验总体流程图

1.1　文法分类

乔姆斯基把文法分成四种类型，即 0 型，1 型，2 型，3 型。

0 型文法定义如下：设 $G = (V_T, V_N, S, P)$ 是一个文法，若$\forall \alpha \to \beta \in P$，$\alpha, \beta \in (V_T \cup V_N)^*$，则 G 是一个 0 型文法。

我们对 0 型文法分别施加以下的第 i 条约束便得到了 i 型文法。

1. G 的任何产生式 $\alpha \to \beta$ 均满足 $|\alpha| \le |\beta|$（其中 $|\alpha|$ 和 $|\beta|$ 分别为 α 和 β 的长度）；仅仅 $S \to \varepsilon$ 例外，但 S 不得出现在任何产生式的右部。
2. G 的任何产生式为 $A \to \beta$，$A \in V_N$，$\beta \in (V_T \cup V_N)^*$。
3. G 的任何产生式为 $A \to \alpha B$ 或 $A \to \alpha$，其中 $\alpha \in V_T^*$，A、$B \in V_N$。

1 型文法也称上下文相关文法。2 型文法也称上下文无关文法。3 型文法也叫做右线性文法。3 型文法还有另外一种形式，叫做左线性文法。由于 3 型文法等价于正规式，所以 3 型文法也可以叫做正规文法。

1.2　巴科斯范式和扩展巴克斯范式

在计算机科学中，巴科斯范式（BNF）是一种表示上下文无关文法技术，通常用于描述计算中使用的语言的语法，例如计算机编程语言，文档格式，指令集和通信协议。 它们可用于需要精确描述语言的地方：例如，官方语言规范，手册和编程语言理论教科书。

现在许多编程语言都可以使用 BNF 来定义，如 C，C++，python 等。而且有些编译器中也使用了 BNF，如著名的 YACC 编译器。

扩展巴克斯范式（EBNF）是对 BNF 进行了扩充。EBNF 描述是 EBNF 规则的无序列表。每一条 EBNF 规则的形式为：LHS ::= RHS。每个 EBNF 规则都包含三个部分：左侧（left－hand side，LHS），::= 和右侧（right-hand side，RHS）。LHS 是 EBNF 规则的名称，::= 表示 "定义为"，RHS 是对此名称的描述。RHS 由规则名称，字符串和控制符组成。

EBNF 中使用的符号定义如表 1 所示：

表 1 EBNF 符号定义表

符号	含义	优先级（数字越大，优先级越低）
< >	尖括号之间的内容用于给出 EBNF 规则的名称	1
::=	定义为	6
( )	用于分组	2
[ ]	表示[和]之间的内容出现或不出现	3
{ }	表示{和}之间的内容可以重复 0 次或若干次	3
‘ ‘	表示字符串常量	1
|	表示或	5

例如表示整数的 EBNF 描述如下：

<符号> ::= '+' | '-'

<非零数字> ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

<数字> ::= '0' | <非零数字>

<整数> ::= '0' | [<符号>] <非零数字> {<数字>}

对于整数来说，要么为 0，要么第一个数字是非零数字，此时整数可以有符号，所以对符号使用了[]，而从第一个非零数字开始又可以有若干个数字。所以整数可以定义为 <整数> ::= '0' | [<符号>] <非零数字> {<数字>}。

2 文法定义

2.1 词法规则定义

首先我们使用 EBNF 来定义词法规则。

词法规则定义见附录 A。

2.2 语法规则定义

根据定义好的词法规则，我们可以使用 EBNF 来定义语法规则G。

语法规则定义见附录 B。

3　将语法规则G（下面简称为文法G）转换成对应的 LL(1)文法

由于使用 EBNF 定义的文法存在 [] 和 {}，我们需要使用 | 和 () 来等价替换掉 [] 和 {}，并对替换后的文法进行化简，将括号去掉 ()。之后再对化简后的文法消除直接左递归、提取公因子，消除冗余，直到将其化为 LL（1）文法。这里我们对其处理的顺序进行了调整，我们将会讨论调整的原因。当然并不是所有的使用 EBNF 定义的文法都可以转换为 LL（1）文法，我们将讨论为什么有些使用 EBNF 定义的文法无法转换为 LL（1）文法。

转换的流程图如图 2 所示：

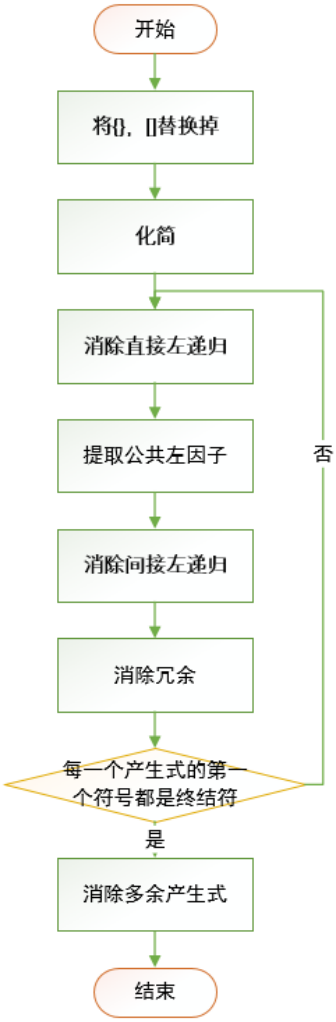

图 2 EBNF 转换为 LL(1)文法的流程图

3.1 替换（消除{}，[]）

我们可以使用如下公式进行替换。

公式 1、

$$A ::= O[P]Q$$

等价于

$$A::= O(P|\varepsilon)Q$$

公式 2、

$$A ::= O\{P\}Q \mid R\{S\}T$$

等价于

$$A ::= OBQ \mid RCT$$

$$B ::= PB \mid \varepsilon$$

$$C ::= SC \mid \varepsilon$$

这样我们就可以将 EBNF 中的[]和{}使用|和()替换掉。

对于文法G，其替换过程如附录 C 所示。

3.2 化简

我们将删除掉文法G中的()，例如有如下产生式：

$$A ::= B(C|D)E|F$$

可以将其转换为：

$$A::= BCE|BDE|F$$

注意这里|的优先级是低于( )的，所以我们先计算( )再计算|。

对于文法G，其化简过程如附录 D 所示。

3.3 消除直接左递归

产生式中不存在直接左递归。

提取公共左因子

当存在公共的最左因子时，FIRST 集合之间会造成两两相交的情况，为了不使 FIRST 集合之间相交，所以需要提取最左因子。

对于产生式：

$$A \rightarrow \delta\beta_1 \mid \delta\beta_2 \mid \delta\beta_3 \mid \cdots \mid \delta\beta_n \mid \gamma_1 \mid \gamma_2 \mid \cdots \mid \gamma_m$$

，就提取成

$$A \rightarrow \delta A' \mid \gamma_1 \mid \gamma_2 \mid \cdots \mid \gamma_m$$

$$A' \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \cdots \mid \beta_n$$

重复这个操作，直到所有的 FIRST 集合两两不相交。

对文法G提取公共左因子的处理见附录 E。

3.4 消除冗余产生式

冗余的产生式对于文法来说不必要，并且还会增加计算的成本，为了减少计算成本，我们需要消除冗余产生式。

假如有如下产生式(还存在其余产生式未给出)

$$D \rightarrow a\ A$$

$$E \rightarrow b\ B$$

$$A \rightarrow \alpha \mid \beta$$

$$B \rightarrow \alpha \mid \beta$$

则我们可以除去 A 和 B 中的任意一个，这里我们消除 B。则产生式可以变为如下形式：

$$D \rightarrow \ a\,A$$

$$E \ \rightarrow \ b\,A$$

$$A \rightarrow \ \alpha \mid \beta$$

以及对于产生式：

$$A \rightarrow \ B$$

我们可以直接将A用B替换，或者B用A替换。

对文法G消除冗余的处理见附录 F。

3.5 消除间接左递归

消除了文法的直接左递归并不意味着消除了文法的左递归，例如文法

$$S \rightarrow Aa \mid a$$

$$A \rightarrow Sb|b$$

则文法还是存在左递归的

$$S \Rightarrow Aa \Rightarrow Sba$$

所以我们设计一种方法来消除文法的间接左递归。

消除规则

1. 若消除过程中出现了直接左递归，就按照直接左递归的方法来消除。
2. 若产生式右部最左的符号是非终结符，且这个非终结符序号大于等于左部非终结符，则暂不处理（后面会处理到）。
3. 若序号小于左部的非终结符，则用之前求到的式子的右部来替换。

消除左递归的算法伪代码描述如下：

Input：

P：由所有非终结符按照某一种排列顺序 $P_1, P_2, \cdots, P_n$构成的序列。

for i in [1,n] do

  for j in [1,i-1] do

    if $P_i \rightarrow P_j\gamma$ and $P_j \rightarrow \delta_1|\delta_2|\cdots|\delta_k$ then

      $P_i \rightarrow P_j\gamma$ 用 $P_i \rightarrow \delta_1\gamma|\delta_2\gamma|\cdots|\delta_k\gamma$ 替换。

    end

  endfor

  消除关于 P_i 的直接左递归

endfor

对文法G消除冗余的处理见附录 G。

3.6 消除关于文法 G 中多余的产生式

重复消除间接左递归,提取公共左因子,消除冗余后文法 G 中存在着大量的多余的产生式。这些多余的产生式对于计算并没有任何的帮助。所以我们可以直接去除掉

例如对于如下产生式

$$S \to aA|bB$$
$$A \to aA|\ \varepsilon$$
$$B \to bBC|\ \varepsilon$$
$$C \to\ bC|\varepsilon$$
$$D \to aA|cCB|bD$$

很显然 D 是多余的，我们可以去除掉D。

化简后的文法G见附录 H。

3.7 调整顺序的原因

消除间接左递归是对若干个不同的终结符来说的，是整体间的关系，而提取公共左因子是对同一个终结符，不同的产生式来说的，是局部的关系。通常我们应该先处理局部关系，再处理整体关系。这样可以避免重复处理相同的规则。简化化简过程。

对于如下产生式：

$$S \to HPe|HP$$
$$P \to HA|A$$
$$A \to cA|\varepsilon$$
$$C \to E|F$$
$$E \to ab|b$$
$$F \to ac|c$$
$$H \to f|g|h|i|\varepsilon$$

显然规则中不存在直接左递归。

若我们按照提取公共左因子，消除间接左递归，消除冗余，消除关于文法 G 中多余的产生式的顺序处理每一个产生式，消除左递归时按照 H,F,E,C,A,P,S来处理有：

$$H \to f|g|h|i|\varepsilon$$
$$F \to ac|c$$
$$E \to ab|b$$
$$C \to ab|b|ac|c$$
$$A \to cA|\varepsilon$$
$$P \to (f|g|h|i|\varepsilon)A|A$$
$$\to fA|gA|hA|iA|A|A$$
$$\to fA|gA|hA|iA|cA|\varepsilon$$
$$S_1 \to HP$$

$$\rightarrow (f|g|h|i|\varepsilon)P$$
$$\rightarrow fP|gP|hP|iP|P$$
$$\rightarrow fP|gP|hP|iP|fA|gA|hA|iA|cA|\varepsilon$$
$$\rightarrow fS_2|gS_2|hS_2|iS_2|cA|\varepsilon$$
$$S_2 \rightarrow P|A$$
$$\rightarrow fA|gA|hA|iA|cA|\varepsilon$$
$$S \rightarrow S_1e|S_1$$
$$\rightarrow fS_2e|gS_2e|hS_2e|iS_2e|cAe|e|fS_2|gS_2|hS_2|iS_2|cA|\varepsilon$$
$$\rightarrow fS_3|gS_3|hS_3|iS_3|cS_4|e|\varepsilon$$
$$S_3 \rightarrow S_2e|S_2$$
$$\rightarrow fAe|gAe|hAe|iAe|cAe|e|fA|gA|hA|iA|cA|\varepsilon$$
$$\rightarrow fS_4|gS_4|hS_4|iS_4|cS_4|e|\varepsilon$$
$$S_4 \rightarrow Ae|A$$
$$\rightarrow cAe|e|cA|\varepsilon$$
$$\rightarrow cS_4|e|\ \varepsilon$$

消除关于文法 G 中多余的产生式，我们便可以得到化简后的规则。

$$S \rightarrow fS_3|gS_3|hS_3|iS_3|cS_4|e|\varepsilon$$
$$S_3 \rightarrow fS_4|gS_4|hS_4|iS_4|cS_4|e|\varepsilon$$
$$S_4 \rightarrow cS_4|e|\ \varepsilon$$

若我们按照消除间接左递归，提取公共左因子，消除冗余，消除关于文法 G 中多余的产生式的顺序处理每一个产生式，则会引入大量的冗余，这里就不展示了。

3.8 EBNF 转换为 LL（1）文法的问题

当然并不是所有的使用 EBNF 定义的文法都可以转换为 LL（1）文法，我们将讨论为什么有些使用 EBNF 定义的文法无法转换为 LL（1）文法。

例如如下用 EBNF 描述的产生式：

$$S \rightarrow s\{a\}a$$

将 {} 进行替换后为

$$S \rightarrow sAa$$
$$A \rightarrow aA|\varepsilon$$

这样的 EBNF 范式是无法转换为 LL（1）文法。

其原因如下：

$$S.first\ =\ \{s\}$$
$$A.first\ =\ \{a,\varepsilon\}$$
$$S.follow\ =\ \{\#\}$$

$$A.follow = \{a\}$$

这个问题是在之前定义的文法中找到的。之前文法中有如下产生式（有些不重要的产生式这里未给出）：

<Start__Y_1__Y_1> ::= <com> <id> <Variable_Def__Y_1> <sem> <Variable_Decl__X_1> <Start__X_1> <Main_Func>

| <lbk> <const_int> <rbk> <Variable_Def__X_1> <sem> <Variable_Decl__X_1> <Start__X_1> <Main_Func>

| <sem> <Variable_Decl__X_1> <Start__X_1> <Main_Func>

| <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>

<Variable_Decl__X_1> ::= <int> <id> <Variable_Def__Y_1> <sem> <Variable_Decl__X_1>

| <float> <id> <Variable_Def__Y_1> <sem> <Variable_Decl__X_1>

| <char> <id> <Variable_Def__Y_1> <sem> <Variable_Decl__X_1>

| <string> <id> <Variable_Def__Y_1> <sem> <Variable_Decl__X_1>

| <eps>

<Start__X_1> ::= <int> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>

| <float> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>

| <char> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>

| <string> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>

| <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>

| <eps>

<Main_Func> ::= <int> <main> <lp> <rp> <lbe> <Compound_Stat> <rbe>

这些产生式不存在左递归且无公因子。但是在最终计算预测分析表的时候却出现了多重定义入口。

再仔细分析后才发现问题所在，是<Variable_Decl__X_1>的 First 集和 Follow 集存在交集，以及<Start__X_1>的 First 集和 Follow 集存在交集。

解决的办法是使<Variable_Decl__X_1>、<Start__X_1>、<Main_Func>的 First 集彼此无交集，所以我在变量的定义前使用了<variable>终结符，以及将<Main_Func>的定义修改为<main> <lp> <rp> <lbe> <Compound_Stat> <rbe>。

4 词法分析

4.1 词法分析的步骤

词法分析程序的步骤如下：

1. 读取源代码程序。
2. 预处理，其目的是去除源代码中的注释（不允许嵌套注释），非字符串中的多余的空格。
3. 词法分析，其目的是对预处理后的字符串进行扫描，产生一个个单词符号。

4. 显示源代码程序中的词法错误。

5. 保存单词符号。

4.2 单词种别码

首先我们先定义词法分析中单词的种别码：

```
#define ERROR -1 // -1 为无法识别的字符标志码
#define EPS 0 // 0 为空串情况 用于理论分析
#define START 1 // 1 为开始情况 用于系统使用
#define CHAR 2 // char
#define CONST 3 // const
#define FLOAT 4 // float
#define IF 5 // if
#define INT 6 // int
#define MAIN 7 // main
#define PRINTF 8 // printf
#define RETURN 9 // return
#define SCANF 10 //scanf
#define STRING 11 // string
#define VARIABLE 12 //variable
#define VOID 13 // void
#define WHILE 14 // while
#define ID 15 // 标识符
#define CONST_CHAR 16 // 字符常量
#define CONST_FLOAT 17 // 浮点数常量
#define CONST_INT 18 // 整型常量
#define CONST_STRING 19 // 字符串常量
#define ADD_SUB 20 // + -
#define MUL_DIV 21 // * /
#define AS 22 // =
#define RELATIONAL_OP 23 // < <= > >= == !=
#define LP 24 // (
#define RP 25 // )
#define LBK 26 // [
#define RBK 27 // ]
#define LBE 28 // {
#define RBE 29 // }
```

#define COM 30 // ,

#define SEM 31 // ;

4.3 状态转换图

定义状态转换图如图 3 所示：

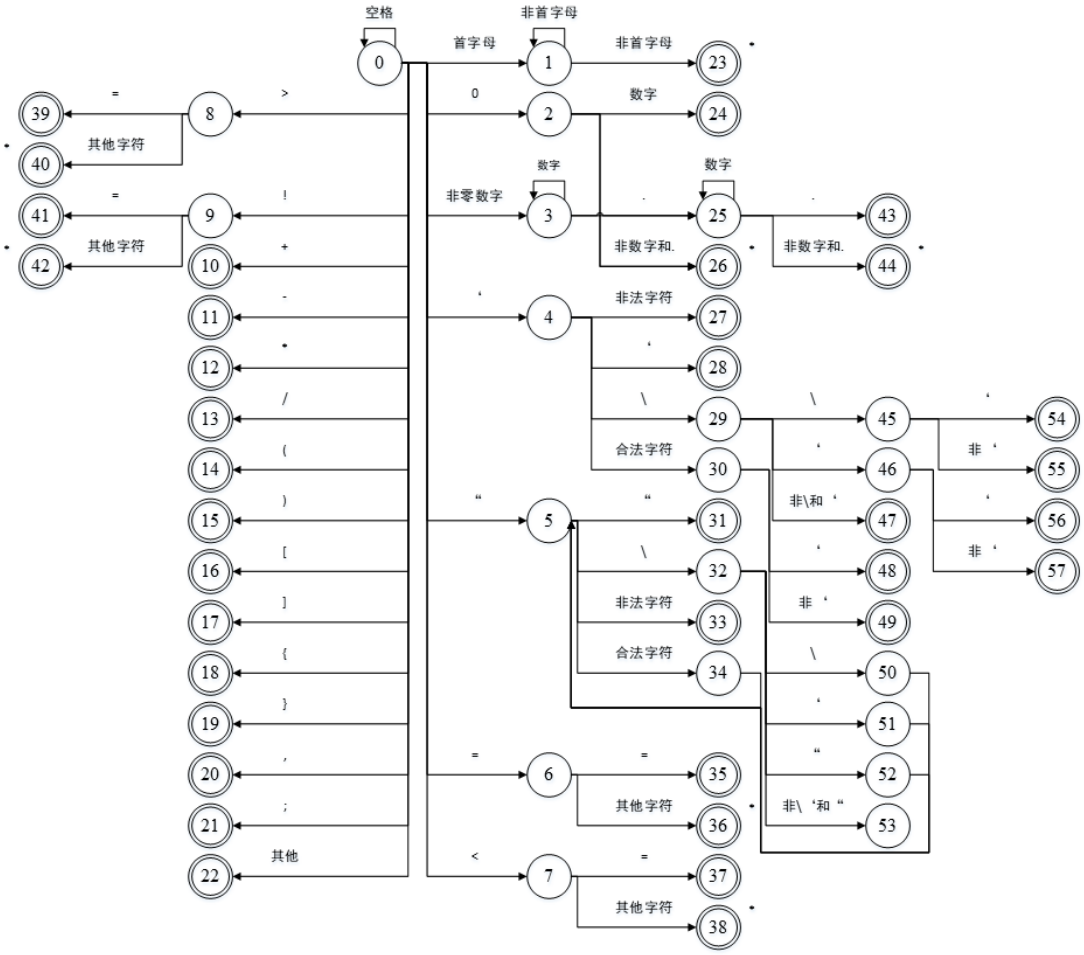

图 3 词法分析状态转换图

根据状态转换图，我们可以实现词法分析的功能。

4.4 读取源代码程序

读取源代码程序实现如下：

```
string scanner_read(string Path) {
    ifstream fin(Path);
    string tmp, ans;
    if (!fin.is_open()) {
        cout << "Error opening file"; exit(1);
    }
    while (!fin.eof()) {
        getline(fin, tmp);
        ans += tmp;
```

```
        }
        return ans;
    }
```

若读取失败则关闭，否则返回源代码字符串。

4.5 预处理

我们需要对源代码中的注释进行删除，并且去除掉源代码中多余的空格，当然，这需要排除掉字符串中的空格，因为这些空格有具体含义。

实现如下：

```
    string preprocess(string & Code) {
        int idx = 0, max_idx = Code.size(), flag = 0;
        string tmp = "";
        while (idx < max_idx) {
            if (Code[idx] == '/' && idx + 1 < max_idx && Code[idx + 1] == '*') {                    // 处理注释
                ++idx;
                while (++idx < max_idx) {
                    if (Code[idx] == '*' && idx + 1 < max_idx && Code[idx + 1] == '/') {      // 结束情况
                        idx += 2;
                        break;
                    }
                }
            }
            else if (flag == 0 && is_space(Code[idx])) {
    // 非字符串中的空格
                tmp += ' ';
                idx++;
                while (idx < max_idx && is_space(Code[idx])) {
                    idx++;
                }
            }
            else if (Code[idx] == '\\' && idx + 1 < max_idx && (Code[idx + 1] == '"' || Code[idx + 1] == '\'')) {       // 字符串中间的 " 和 '
                tmp += Code[idx];
                tmp += Code[idx + 1];
```

```
            idx += 2;
        }
        else if ((Code[idx] == '"' || Code[idx] == '\'')) {                    // 字
符串开始或结束
            flag = (flag + 1) % 2;
            tmp += Code[idx++];
        }
        else {
    // 非特殊情况
            tmp += Code[idx++];
        }
    }
    return tmp;
}
```

4.6 词法分析

词法分析函数实现如下：

```
Token Scanner() {
    string tmp = "";
    while (true) {
        char ch = GetCh();
        if (is_space(ch)) {} // 空格
        else if (is_initial(ch)) {                                    // 关键字 和
标识符
            /*
            <标识符> ::= <首字母> {<非首字母>}
            <首字母> ::= '_' | <小写字母> | <大写字母>
            <非首字母> ::= <首字母> | <数字>
            */
            tmp = ch;
            while (ch = GetCh()) {
                if (is_non_initial(ch)) {
                    tmp += ch;
                }
                else {
                    Retract();
```

```
                return Token{ keyword_or_identifier(tmp) , tmp };      // 关键字 或
标识符
            }
        }
    }
    else if (is_num(ch)) {                                              // 处理
整型常量 和 浮点型常量 CONST_INT and CONST_FLOAT
        /*
        <整型常量> ::= '0' | <非零数字> {<数字>}
        <浮点常量> ::= <整型常量> '.' <数字> {<数字>}
        1. d_0 d
        2. d_0 .
        3. d_0 c
        4. 0 d          非法情况
        5. 0 .
        6. 0 c
        7. . .          非法情况
        */
        tmp = ch;
        bool is_non_zero = is_non_zero_num(ch);
        bool flag = false; // false 为整数, true 为浮点数
        while (ch = GetCh()) {
            if (is_num(ch)) {                           // case 1, 4
                if (!flag && !is_non_zero) {                // case 4
                    ProcError();
                    return Token{ ERROR, "-" };
                }
                else {                                  // case 1
                    tmp += ch;
                }
            }
            else if (ch == '.') {               // case 2, 5, 7
                if (flag) {                             // case 7
                    ProcError();
                    return Token{ ERROR, "-" };
```

```
                }
                else {                                  // case 2, 5
                    flag = true;
                    is_non_zero = false;
                    tmp += ch;
                }
            }
            else {                                      // case 3
                Retract();
                return    Token{ flag ? CONST_FLOAT : CONST_INT , tmp };
            }
        }
    }
    else if (ch == '\'') {                                          // 处理字符常量 CONST_CHAR
        tmp = "";
        /*
        已经读入了一个单引号
        0. '非法字符
        1. '
        2. '\非法转义
        3. '\'
        4. '\\
        5. 'c

        在 case 2, 3, 4, 5 基础上 判断 e'
        6. ee' // 多个字符
        7. e'  // 合法情况
        */
        ch = GetCh();
        if (!is_legal_ch(ch)) {      // case 0
            ProcError();
            return Token{ ERROR, "-" };
        }
        else if (ch == '\'') {   // case 1
```

```
                tmp = "";
            }
            else {                          // case 2, 3, 4, 5, 6, 7
                if (ch == '\\') {                   // case 2, 3, 4
                    ch = GetCh();
                    if (ch != '\'' && ch != '\\') {     // case 2
                        ProcError();
                        return Token{ ERROR, "-" };
                    }
                    else if (ch == '\'') {          // case 3
                        tmp = "\'";
                    }
                    else {                                  // case 4
                        tmp = "\\";
                    }
                }
                else {                                      // case 5
                    tmp += ch;
                }
                ch = GetCh();
                if (ch != '\'') {                   // case 6
                    ProcError();
                    return Token{ ERROR, "-" };
                }
                else {                                  // case 7
                    return Token{ CONST_CHAR , tmp };
                }
            }
        }
        else if (ch == '"') {                                       // 处理字符串常量 CONST_STRING
            tmp = "";
            /*
            已经读入了一个双引号
            0   "..."                   结束情况
```

1. "...\非法转义　　非法情况

2. "...\"

3. "...\'

4. "...\\

5. "...c

6. "...非法字符　　非法情况

7. "...eof　　非法情况

*/

```
while (Code_Idx < Code.size() && (ch = GetCh())) { // case 0, 1, 2, 3, 4, 5, 6
    if (ch == '"') {                    // case 0
        return Token{ CONST_STRING, tmp };
    }
    else if (ch == '\\') {              // case 1, 2, 3, 4
        ch = GetCh();
        if (ch == '"') {                // case 2
            tmp += "\"";
        }
        else if (ch == '\'') {          // case 3
            tmp += '\'';
        }
        else if (ch == '\\') {          // case 4
            tmp += '\\';
        }
        else {                          // case 1
            ProcError();
            return Token{ ERROR, "-" };
        }
    }
    else if (is_legal_ch(ch)) {         // case 5
        tmp += ch;
    }
    else {                              // case 6
        ProcError();
        return Token{ ERROR, "-" };
    }
```

```
        }
    }
    else if (ch == '=') {                       // = ==
        if (GetCh() == '=') {                   // ==
            return Token{ RELATIONAL_OP ,"==" };
        }
        else {                                          // =
            Retract();
            return Token{ AS ,"-" };
        }
    }
    else if (ch == '<') {                       // < <=
        if (GetCh() == '=') {                   // <=
            return Token{ RELATIONAL_OP ,"<=" };
        }
        else {                                          // <
            Retract();
            return Token{ RELATIONAL_OP ,"<" };
        }
    }
    else if (ch == '>') {                           // > >=
        if (GetCh() == '=') {                   // >=
            return Token{ RELATIONAL_OP ,">=" };
        }
        else {                                          // >
            Retract();
            return Token{ RELATIONAL_OP ,">" };
        }
    }
    else if (ch == '!') {                       // != and ERROR
        if (GetCh() == '=') {                           // !=
            return Token{ RELATIONAL_OP ,"!=" };
        }
        else {                                          // ERROR
            Retract();
```

```
                ProcError();
                return Token{ ERROR, "-" };
            }
        }
        else if (ch == '+') { return Token{ ADD_SUB, "+" }; }          // +
        else if (ch == '-') { return Token{ ADD_SUB, "-" }; }          // -
        else if (ch == '*') { return Token{ MUL_DIV, "*" }; }          // *
        else if (ch == '/') { return Token{ MUL_DIV, "/" }; }          // /
        else if (ch == '(') { return Token{ LP, "-" }; }               // (
        else if (ch == ')') { return Token{ RP, "-" }; }               // )
        else if (ch == '[') { return Token{ LBK, "-" }; }              // [
        else if (ch == ']') { return Token{ RBK, "-" }; }              // ]
        else if (ch == '{') { return Token{ LBE, "-" }; }              // {
        else if (ch == '}') { return Token{ RBE, "-" }; }              // }
        else if (ch == ',') { return Token{ COM, "-" }; }              // ,
        else if (ch == ';') { return Token{ SEM, "-" }; }              // ;
        else {
            ProcError();
            return Token{ ERROR, "-" };
        }
    }
}
```

4.7 显示源代码程序中的词法错误

若词法分析错误，则种别码会显示-1。词法分析的错误在词法分析程序中已经处理过了。

4.8 保存单词符号

保存词法分析的结果的代码如下：

```
void scanner_write(string Path) {
    ofstream fout(Path, ios::out);
    for (int i = 0; i < tokens.size(); i++) {
        fout << "Token " << i << ": " << tokens[i].type << " " << tokens[i].value << endl;
    }
    fout.close();
}
```

5 语法分析程序

语法分析程序的步骤如下：

1. 根据文法得到 First 集。
2. 根据文法和 First 集得到 Follow 集。
3. 根据文法、First 集、Follow 集得到预测分析表。
4. 根据预测分析表和词法分析得到的单词符号进行分析，显示分析情况。
5. 显示源代码程序中的语法错误。
6. 保存相关记录。

5.1 计算文法的 $First$ 集合

5.1.1 计算过程描述

对每一个文法符号 $X \in (V_T \cup V_N)^*$，只需要连续使用如下规则直到每个 $First$ 集合不在增大为止。

1. 若 $X \in V_T$ ，则 $First(X) = \{X\}$。
2. 若 $X \in V_N$，且有产生式 $X \to a\cdots$ ，则把 a 加入到 $First(X)$ 中。
3. 若 $X \to \varepsilon$ 也是一条产生式，则把 ε 加入到 $First(X)$ 中。
4. 若 $X \to Y\cdots$ 是一个产生式且 $Y \in V_N$，则把 $First(Y)$ 中的所有的非 ε 元素都加到 $First(X)$ 中。
5. 若 $X \to Y_1Y_2\cdots Y_k$ 是一个产生式，Y_1，…，Y_{i-1} 都是非终结符，而且 $\forall j$ 满足 $1 \le j \le i-1$，$First(j)$ 都含有 ε（即 $Y_1\cdots Y_{i-1} \Rightarrow \varepsilon$），则把 $First(Y_i)$ 中的所有的非 ε 元素都加到 $First(X)$ 中。特别是若所有的 $First(Y_j)$ 均含有 ε，$j = 1,2,\cdots,k$，则把 ε加到 $First(X)$ 中。

5.1.2 优化

下面我们对上述求解过程进行优化，并将其用算法伪代码形式描述。

对于第 2-5 条，我们假设产生式为 $X \to Y_1Y_2\cdots Y_k$，且 $Y_1Y_2\cdots Y_i \Rightarrow \varepsilon$。则对于 Y_i 来说，将 $First(Y_i) \setminus \{\varepsilon\}$ 加入到 $First(X)$，若 $\varepsilon \in First(Y_i)$，则将 $i+1$。若 i 等于 k，则将 ε 加入到 $First(X)$ 中。

5.1.3 算法伪代码描述

First 集合计算过程算法伪代码描述如下：

Input：

P：由所有产生式构成的集合。

P_i：第i 条产生式为 $X_i \to Y_{i1}Y_{i2}\cdots Y_{it}$。

a_1, a_2,…, a_k 为所有终结符。

$A_1, A_2, \cdots, A_n$ 为所有非终结符。

Initialize：

$flag = false$

$a_1.first = \emptyset, a_2.first = \emptyset, \cdots, a_k.first = \emptyset$

$A_1.first = \emptyset, A_2.first = \emptyset, \cdots, A_n.first = \emptyset$

for $i \in [1, k]$ do

$a_i.first = \{a_i\}$

endfor

do

$flag = false$

foreach $P_i \in P$ do

for $j \in [1, t]$ do

if $X_i.first \setminus \{\varepsilon\} \neq Y_{ij}.first \setminus \{\varepsilon\}$ then

$flag = true$

$X_i.first \cup= Y_{ij}.first \setminus \{\varepsilon\}$

end

if $\varepsilon \notin Y_{ij}.first$ then

break

end

endfor

if $j == t$ then

$X_i.first \cup= \{\varepsilon\}$

end

end

while $flag == true$

5.1.4 C++实现

First 集合计算 C++实现如下:

```
void get_first() {
    bool flag;
    do {
        flag = false;
        for (auto & prod : prods) {
            int i = 0;
            for (; i < prod.right.size(); i++) {
                if (nodes[prod.left].first_insert_first(nodes[prod.right[i]])) {
                    flag = true;
                }
```

```
                    if (!nodes[prod.right[i]].first_empty) {
                        break;
                    }
                }
                if (i == prod.right.size()) {
                    nodes[prod.left].first_empty = true;
                }
            }
        } while (flag);
    }
```

5.2 计算文法的 $\boldsymbol{Follow}$ 集合

5.2.1 计算过程描述

对文法 G 的每个非终结符 A 构造 $Follow(A)$ 的方法为使用如下规则直到每一个 $Follow$ 不再增大为止。

1. 若 A 是开始符号，置 {# 于 $Follow(A)$ 中。
2. 若 $A \to \alpha B\beta$ 是一个产生式，则把 $First(\beta) \setminus \{\varepsilon\}$ 加入到 $Follow(B)$ 中；
3. 若 $A \to \alpha B$ 是一个产生式，或 $A \to \alpha B\beta$ 是一个产生式且 $\beta \Rightarrow \varepsilon$（即 $\varepsilon \in First(\beta)$），则把 $Follow(A)$ 加入到 $Follow(B)$ 中。

5.2.2 优化

对计算 $Follow$ 集方法的优化

我们对计算 $Follow$ 集的规则 2 和 3 进行重组，改为如下形式：

若 $A \to \alpha B$ 是一个产生式，则把 $Follow(A)$ 加入到 $Follow(B)$ 中。

若 $A \to \alpha B\beta$ 是一个产生式，则把 $First(\beta) \setminus \{\varepsilon\}$ 加入到 $Follow(B)$ 中；若 $\varepsilon \in First(\beta)$，则把 $Follow(A)$ 加入到 $Follow(B)$ 中。

注意到，计算 B 的 $Follow$ 集，只与 A 或 β 有关，而与 α 无关。

则对于产生式 $X \to Y_1Y_2 \cdots Y_t$，计算 $Follow(Y_i)$ 只与 $X, Y_{i+1}, \cdots, Y_t$ 有关。

设

$$g(x) = \begin{cases} 1, x为true \\ 0, x为false \end{cases}$$

我们可以得到

$$\begin{aligned} & Follow(Y_i) \\ \cup= & First(Y_{i+1} \cdots Y_t) \setminus \{\varepsilon\} + g\big(\varepsilon \in First(Y_{i+1} \cdots Y_t)\big) \times Follow(A) \\ \cup= & First(Y_{i+1}) \setminus \{\varepsilon\} + g(\varepsilon \in Y_{i+1}) \times (First(Y_{i+2} \cdots Y_t) \setminus \{\varepsilon\}) + g(\varepsilon \\ & \in First(Y_{i+1} \cdots Y_t)) \times Follow(A) \end{aligned}$$

$$\cup= First(Y_{i+1}) \setminus \{\varepsilon\} + g(\varepsilon \in Y_{i+1}) \times (First(Y_{i+2} \cdots Y_t) \setminus \{\varepsilon\})$$
$$+ g(\varepsilon \in Y_{i+1}) g\big(\varepsilon \in First(Y_{i+2} \cdots Y_t)\big) \times Follow(A)$$
$$\cup= First(Y_{i+1}) \setminus \{\varepsilon\} + g(\varepsilon \in Y_{i+1}) \times (First(Y_{i+1} \cdots Y_t) \setminus \{\varepsilon\} + g(\varepsilon$$
$$\in First(Y_{i+2} \cdots Y_t)) \times Follow(A))$$
$$\cup= First(Y_i) \setminus \{\varepsilon\} + g(\varepsilon \in Y_{i+1}) \times Follow(Y_{i+1})$$

以上假设 Y_i 和 Y_{i+1} 均 $\in V_N$。而终结符没有 $Follow$ 集合。

5.2.3 算法伪代码描述

Follow 集合计算过程算法伪代码描述如下：

Input：

P：由所有产生式构成的集合。

P_i：第 i 条产生式为 $X_i \rightarrow Y_{i1}Y_{i2} \cdots Y_{it}$。

$a_1, a_2, \cdots, a_k$ 为所有终结符。

$A_1, A_2, \cdots, A\_n$ 为所有非终结符。

S 为开始符号。

$a_1.first, a_2.first, \cdots, a_k.first$ 为所有终结符的 $first$ 集合。

$A_1.first, A_2.first, \cdots, A_n.first$ 为所有非终结符的 $first$ 集合。

Initialize：

$flag = false$

$A_1.follow = \emptyset, A_2.follow = \emptyset, \cdots, A_n.follow = \emptyset$

$S = \{\#\}$

do

$flag = false$

foreach $P_i \in P$ do

$j = t$

if $is\_V_T(Y_{ij})$ then

if $Y_{ij}.follow \neq X.follow$ then

$flag = true$

$Y_{ij}.follow \cup= X.follow$

end

end

for $j \in [t, 1]$ do

if $is\_V_T(Y_{ij})$ then

continue

end

if $Y_{i,j-1}.follow \neq Y_{ij}.first \setminus \{\varepsilon\}$ then

$flag = true$

$Y_{i,j-1}follow \cup= Y_{ij}.first \setminus \{\varepsilon\}$

end

if $\varepsilon \in Y_{ij}.first$ then

if $Y_{i,j-1}.follow \neq Y_{ij}.follow$ then

$flag = true$

$Y_{i,j-1}.follow \cup= Y_{ij}.follow$

end

end

endfor

end

while $flag == true$

5.2.4 C++实现

Follow 集合计算 C++实现：

```
void get_follow() {
    nodes[VN_Start_Idx].follow.insert(VT_Start_Idx);
    bool flag;
    do {
        flag = false;
        for (auto & prod : prods) {
            int idx = prod.right.size() - 1;
            if (!nodes[prod.right[idx]].is_vt) {
                if (nodes[prod.right[idx]].follow_insert_follow(nodes[prod.left])) {
                    flag = true;
                }
            }
            for (; idx > 0; idx--) {
                if (nodes[prod.right[idx - 1]].is_vt) {
                    continue;
                }
                if (nodes[prod.right[idx - 1]].first_insert_follow(nodes[prod.right[idx]])) {
                    flag = true;
                }
                if (nodes[prod.right[idx]].first_empty) {
```

```
                                    if (nodes[prod.right[idx-
1]].follow_insert_follow(nodes[prod.right[idx]])) {
                                        flag = true;
                                    }
                                }
                            }
                        }
                    } while (flag);
                }
```

5.3 构建预测分析表

5.3.1 计算过程描述

预测分析表是一个 M[A，a]形式的矩阵。其中，A 为非终结符，a 是终结符。矩阵元素 M[A，a]中存放着一条关于 A 的产生式，指出当 A 面临输入符号 a 时所应采用的候选。M[A，a]中也可能存放一个“出错标志”，指出 A 根本不该面临输入符号 a。

对文法 G 的每一个产生式 $A \to \alpha$ 执行第 2 至第 3 步。

对每个终结符 $a \in \alpha.first$，把 $A \to \alpha$ 加至 $M[A,a]$ 中。

若 $\varepsilon \in \alpha.first$，则对任何 $b \in A.follow$ 把 $A \to \alpha$ 加至 $M[A,b]$ 中。

把所有无定义的 $M[A,a]$ 标上 "错误标志"。

5.3.2 优化

假设产生式为 $X \to Y_1Y_2 \cdots Y_k$，且 $Y_1Y_2 \cdots Y_{i-1} \Rightarrow \varepsilon$。则对于 Y_i 来说，将 $M[X,a]$ 置为 $X \to Y_1Y_2 \cdots Y_k$，其中 $a \in Y_i.first$。若 $\varepsilon \notin Y_i$，则不能执行第 3 步，则结束，否则将 $i+1$。若 i 等于 k，则将 $M[X,b]$ 置为 $X \to Y_1Y_2 \cdots Y_k$，其中 $b \in A.follow$。

5.3.3 算法伪代码描述

预测分析表算法伪代码实现如下：

Input：

P：由所有产生式构成的集合。

P_i：第 i 条产生式为 $X_i \to Y_{i1}Y_{i2} \cdots Y_{it}$。

a_1，a_2，$\cdots$，a_k 为所有终结符。

$A_1, A_2, \cdots, A_n$ 为所有非终结符。

$a_1.first, a_2.first, \cdots, a_k.first$ 为所有终结符的 $first$ 集合。

$A_1.first, A_2.first, \cdots, A_n.first$ 为所有非终结符的 $first$ 集合。

$A_1.follow, A_2.follow, \cdots, A_n.follow$ 为所有非终结符的 $follow$ 集合。

Initialize：

for $x \in [1,n]$ do

for $y \in [1,k]$ do

$M[x, y] = -1$

endfor

endfor

foreach $P_i \in P$ do

for $j \in [1, t]$ do

foreach $a \in Y_{ij}.first$ do

$M[X, a] = X_i \rightarrow Y_{i1}Y_{i2} \cdots Y_{it}$

end

if $\varepsilon \notin Y_{ij}.first$ then

break

end

endfor

if $j == t + 1$ then

foreach $b \in X_i.follow$ do

$M[X, b] = X_i \rightarrow Y_{i1}Y_{i2} \cdots Y_{it}$

end

end

end

5.3.4 C++实现

预测分析表计算 C++实现如下：

```
void get_table() {
    for (int i = 0, idx; i < prods.size(); i++) {
        for (idx = 0; idx < prods[i].right.size(); idx++) {
            for (auto & first : nodes[prods[i].right[idx]].first) {
                table[prods[i].left - VN_Start_Idx][first] = i;
            }
            if (!nodes[prods[i].right[idx]].first_empty) {
                break;
            }
        }
        if (idx == prods[i].right.size()) {
            for (auto & follow : nodes[prods[i].left].follow) {
                table[prods[i].left - VN_Start_Idx][follow] = i;
            }
        }
```

}

}

5.4 预测分析程序的总控程序

5.4.1 计算过程描述

栈 Stack 用于存放文法符号。分析开始时，栈底先放一个#，然后，放进文法开始符号。同时，假定输入串之后也总有一个#，标志输入串结束。

预测分析程序的总控程序在任何时候都是按 Stack 栈顶符号X和当前的输入符号a行事的。对于任何(X,a)，总控程序每次都执行下述三种可能的动作之一：

1. 若$X=a='\#'$，则宣布分析成功，停止分析过程。
2. 若$X=a\neq'\#'$，则把X从 Stack 栈顶逐出，让a指向下一个输入符号。
3. 若X是一个非终结符，则查看分析表M。若$M[A,a]$中存放着关于X的一个产生式，那么，首先把X逐出 Stack 栈顶，然后，把产生式的右部符号串按反序一一推进 Stack 栈（若右部符号为ε，则意味不推什么东西进栈）。若$M[A,a]$中存放着“出错标志”，则调用出错诊察程序 ERROR。

5.4.2 算法伪代码描述

预测分析程序的总控程序伪代码描述如下：

Input：

P：由所有产生式构成的集合。

P_i：第 i 条产生式为 $X_i \rightarrow Y_{i1}Y_{i2}\cdots Y_{it}$。

a_1，a_2,⋯，a_k 为所有终结符。

$A_1,A_2,\cdots,A_n$ 为所有非终结符。

$a_1.first,a_2.first,\cdots,a_k.first$ 为所有终结符的 $first$ 集合。

$A_1.first,A_2.first,\cdots,A_n.first$ 为所有非终结符的 $first$ 集合。

$A_1.follow,A_2.follow,\cdots,A_n.follow$ 为所有非终结符的 $follow$ 集合。

M：预测分析表，n行k列。

Input_Str：输入串。

Idx: 输入串对应

Initialize：

```
Stack.push('#')          // 推入 '#'
Stack.push(Start)        // 推入文法开始符号
Input_Str +='#'          // 输入串末尾加入开始符号
```

```
While !stack.empty() do
    X = Stack.top()
    Stack.pop()
    if is_V_T(X) then
```

if $X == a$ then

a = Input_Str[Idx++]

else

ERROR

end

else if $X ==$ '#' then

if $X == a$ then

FINISH

else

ERROR

end

else if $M[A, a] == P_i$ then

for $j \in [t, 1]$ do

if $Y_{ij} \neq \varepsilon$ then

stack.push(Y_{ij})

end

endfor

else

ERROR

end

end

5.4.3 C++实现

预测分析程序的总控程序实现如下：

```
bool analyze(vector<int> & v, string Path) {
    vector<int> stack = { VT_Start_Idx ,VN_Start_Idx };
    v.push_back(VT_Start_Idx);
    int idx = 0;
    Prod tmp = Prod();
    show_analyze(stack, v, idx, tmp);
    write_analyze(stack, v, idx, tmp, Path);
    while (!stack.empty()) {
        int cur = stack[stack.size() - 1];
        stack.pop_back();
        if (nodes[cur].is_vt && cur != VT_Start_Idx) {
            if (cur == v[idx]) {
```

```
                idx++;
                show_analyze(stack, v, idx, tmp);
                write_analyze(stack, v, idx, tmp, Path);
            }
            else {
                return false;
            }
        }
        else if (cur == VT_Start_Idx) {
            if (cur == v[idx]) {
                return true;
            }
            else {
                return false;
            }
        }
        else if (table[cur - VN_Start_Idx][v[idx]] != -1) {
            auto & right = prods[table[cur - VN_Start_Idx][v[idx]]].right;
            for (auto it = right.rbegin(); it != right.rend(); ++it) {
                if (*it != VT_Empty_Idx) {
                    stack.push_back(*it);
                }
            }
            show_analyze(stack, v, idx, prods[table[cur - VN_Start_Idx][v[idx]]]);
            write_analyze(stack, v, idx, prods[table[cur - VN_Start_Idx][v[idx]]], Path);
        }
        else {
            return false;
        }
    }
    return true;
}
```

预测分析程序输出：

```
void show_analyze(vector<int> & stack, vector<int> & v, int idx, Prod & prod) {
    cout << "stack:" << endl;
```

```
    for (auto it = stack.begin(); it != stack.end(); it++) {
        cout << g(*it) << " ";
    }
    cout << endl << "input_string:" << endl;
    for (auto it = v.begin() + idx; it != v.end(); it++) {
        cout << g(*it) << " ";
    }
    cout << endl << "prod:" << prod << endl;
}
```

预测分析程序结果保存：

```
void write_analyze(vector<int> & stack, vector<int> & v, int idx, Prod & prod, string Path) {
    ofstream fout(Path, ios::app);
    fout << "stack:" << endl;
    for (auto it = stack.begin(); it != stack.end(); it++) {
        fout << g(*it) << " ";
    }
    fout << endl << "input_string:" << endl;
    for (auto it = v.begin() + idx; it != v.end(); it++) {
        fout << g(*it) << " ";
    }
    fout << endl << "prod:" << prod << endl;
    fout.close();
}
```

主函数：

```
int main() {
    string BasePath = "C:/Users/19799/Desktop/编译原理课设/测试用例/"; //  文件夹路径
    string Scanner_Path = "test1"; //  词法分析输入路径
    string Parser_Path = "test15"; //  语法分析输入路径

    /*词法分析*/
    Code = scanner_read(BasePath + Scanner_Path + ".in");
    cout << Code << endl;
    Code = preprocess(Code);
    cout << Code << endl;
    vector<int> v = getToken();
```

```
        scanner_show();
        scanner_write(BasePath + Scanner_Path + ".out");

        /*语法分析*/
        if (!Error_Times) {
            parser_read(BasePath + Parser_Path + ".in");
            get_first();
            get_follow();
            parser_write(BasePath + Parser_Path + ".out");
            get_table();
            cout << (analyze(v, BasePath + Parser_Path + ".out") ? "语法正确" : "语法错误") <<
endl;
        }
        else {
            cout << "有词法错误" << endl;
        }
        system("pause");
        return 0;
    }
```

6 测试

6.1 测试用例 1：

该源代码不存在词法与语法问题。测试用例输入文件名称为 test1.in，输出为 test1.out 和 test15.out。

6.1.1 正确输入 1：

```
/*常量定义*/
const char C1 = 'a', C2 = 'b', C3 = '3';
const float F1 = 1.2, F2 = 1.5, F3 = 1.8;
const int I1 = 10, I2 = 20, I3 = 30;
const string S1 = "hello world", S2 = "20171344137", S3 = "20171344106";
/*变量定义*/
variable char c1, c2[10], c3, c4[20], c5;
variable float f1[30], f2, f3[40], f4, f5[50];
variable int i1, i2[60], i3, i4[70], i5;
variable string s1[80], s2, s3[90], s4, s5[100];
/*函数定义*/
```

```
float fun1(int a, float b){
    return 1.1 * 2.2;
}
int fun2(char c){
    return 1 + 1 * (1 / 1 - 1);
}
void fun3(int a, float b, char c){
    fun1(a, b);
    fun2(c);
}
/*主函数*/
main() {
    scanf(c1, c3, c5, f2, f4, i1, i3, i5, s2, s4);
    printf("Hello World", i1 + i3 + i5);
    printf("Hello World");
    printf(i1 + i3 + i5);
    while(i5 + i3 == 1) {
        if(i3 < i5) {
            i5 = i3 - 10;
        }
        if(i3 > 15 + 10){
            i3 = i5 - 10;
        }
    }
}
```

6.1.2 正确输出 1：

由于词法分析与语法分析结果过长，具体结果见文件 test1.out。

6.2 测试用例 2：

该测试用例中存在词法问题，具体问题在测试用例中已列出。测试用例输入文件名称为 test_error_1.in，输出为 test_error_1.out 和 test15.out。

6.2.1 错误输入 1：

```
/*词法分析错误示例*/
/* 1. 注释使用方式错误
   2. 前导 0
   3. 小数中出现多个小数点
```

4. 非法转义
5. 非法字符
6. 字符常量长度超过 1
7. 单引号不匹配
8. 双引号不匹配
9. 单独出现感叹号(文法中未定义非)
10. 出现未定义的符号

```
*/
//      /*/      /*/*/
00       01
0.0.0
'\h'       '\e'       '\p'
'我'
'hello world'
's''
! @ # $ % ^ &
"ss"
```

6.2.2 错误输出 1：

由于词法分析与语法分析结果过长，具体结果见文件 test_error_1.out。

6.3 测试用例三

该测试用例中不存在词法问题，但是存在语法问题。测试用例输入文件名称为 test_error_2.in，输出为 test_error_2.out 和 test15.out。

6.3.1 错误输入 2：

```
/*词法分析错误示例*/
variable int i1, i2, i3, i4, i5;
main() {
    printf(i1 + i3 + i5, i2 + i4 + i6); /*不存在这类输出函数的定义*/
    return 0;
}
```

6.3.2 错误输出 2：

由于词法分析与语法分析结果过长，具体结果见文件 test_error_2.out。

7 总结

通过这次实验，我们掌握了 EBNF 的使用，并使用它来定义了简单的类似 C 语言的语法规则，当然此语法规则并不像标准的 C 语言语法一样完美，但是可以用于简单的源代码的语法分析。同时，我们还对原来的词法分析过程进行了重写，原来的词法规则并不完美，经过

了这次的修改，对其不完善的地方进行了改进。同时我们还使用了自顶向下的语法分析实现了预测分析程序。对源代码进行了语法分析。我们还对 First 集合、Follow 集合、预测分析表的计算进行了优化，使其效率更高。总体来说，这次实验对我们来说，收获很大。

8 讨论

1. 是否所有的 EBNF 都可以转换为 LL（1）文法。
2. 是否所有的 EBNF 都不可以转换为 LL（1）文法。
3. 哪样的 EBNF 可以转换为 LL（1）文法。
4. EBNF 转换为 LL（1）文法会产生的问题。
5. 如何将 EBNF 转换为 LL（1）文法。
6. 消除间接左递归，提取公共左因子，消除冗余，消除多余的产生式的顺序问题。
7. 词法分析预处理中注释嵌套，消除空格的问题。
8. 语法分析 First 集合，Follow 集合，预测分析表的求解和优化问题。
9. 词法分析的错误处理问题。
10. 语法分析的错误处理问题。

其中问题 3 和 10 还未解决。其余问题均在前面有所论述。

在语法分析中，当遇到错误后，程序便会输出语法分析错误，并结束，这样是远远不够的，在之后的修改中会针对这一问题进行修改。同时会对词法和语法分析程序进行优化，使得其效率更高。

参考文献

[1] Harrison M (1978) Introduction to formal language theory. Addison Wesley, Reading

[2] Hopcroft J, Ullman J (1969) Formal languages and their relation to automata. Addison-Wesley,Reading

[3] BNF paradigm representation of C

[4] Article "EBNF: A Notation to Describe Syntax (PDF)" by Richard E. Pattis describing the functions and syntax of EBNF

[5] ISO/IEC 14977 : 1996(E)

[6] http://www.cs.man.ac.uk/~pjj/bnf/c_syntax.bnf

[i]

[i] 尾注

附录A

EBNF 定义的词法规则如下：

<non_zero_digit> ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

<digit> ::= '0' | <non_zero_digit>

<lower_letter> ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'

<upper_letter> ::= 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'

<special_symbol> ::= ' ' | '"' | '!' | '#' | '$' | '%' | '&' | '\'' | '(' | ')' | '*' | '+' | ',' | '-' | '.' | '/' | ':' | ';' | '<' | '=' | '>' | '?' | '@' | '[' | '\\' | ']' | '^' | '_' | '`' | '{' | '|' | '}' | '~'

<character> ::= <digit> | <lower_letter> | <upper_letter> | <special_symbol>

<Initial> ::= '_' | <lower_letter> | <upper_letter>

<non_Initial> ::= <Initial> | <digit>

<eps> ::= epsilon # for Derivation

<start> ::= # # for system

<char> ::= 'char'

<const> ::= 'const'

<float> ::= 'float'

<if> ::= 'if'

<int> ::= 'int'

<main> ::= 'main'

<printf> ::= 'printf'

<return> ::= 'return'

<scanf> ::= 'scanf'

<string> ::= 'string'

<variable> ::= 'variable'

<void> ::= 'void'

<while> ::= 'while'

<id> ::= <Initial> {<non_Initial>}

<const_ch> ::= '\'' [<character>] '\''

<const_flt> ::= <const_int> '.' <digit> {<digit>}

<const_int> ::= '0' | <non_zero_digit> {<digit>}

<const_str> ::= '"' {<character>} '"'

<add_sub> ::= '+' | '-'

<mul_div> ::= '*' | '/'

<as> ::= '='

<relational_op> ::= '<'|'<='|'>'|'>='|'=='|'!='

<lp> ::= '('

<rp> ::= ')'

<lbk> ::= '['

<rbk> ::= ']'

<lbe> ::= '{'

<rbe> ::= '}'

<com> ::= ','

<sem> ::= ';'

附录B

根据词法规则，使用 EBNF 定义语法规则如下：

$G = (V_N, V_T, S, P)$，

V_N定义如下：

{<Assign_Stat>, <Cond>, <Constant_Decl>, <Constant_Def>, <Cond_Stat>, <Compound_Stat>, <Decl_Head>, <Expr>, <Func_Call_Stat_Non>, <Factor>, <Func_Def_Val>, <Func_Def_Non>, <Func_Call_Stat_Val>, <Item>, <Loop_Stat>, <Main_Func>, <Param>, <Print_Stat>, <Ret_Stat>, <Start>, <Stat>, <Sgn_Int>, <Stats>, <Sgn_Flt>, <Scanf_Stat>, <Type_id>, <Value_Params>, <Variable_Def>, <Variable_Decl>}

V_T定义如下：

{<eps>, <start>, <char>, <const>, <float>, <if>, <int>, <main>, <printf>, <return>, <scanf>, <string>, <void>, <while>, <variable>, <id>, <const_ch>, <const_flt>, <const_int>, <const_str>, <add_sub>, <mul_div>, <as>, <relational_op>, <lp>, <rp>, <lbk>, <rbk>, <lbe>, <rbe>, <com>, <sem>}

S定义如下：

<Start>

P定义如下：

<Start> ::= [<Constant_Decl>] [<Variable_Decl>] { <Func_Def_Val> | <Func_Def_Non> }
<Main_Func>

<Constant_Decl> ::= <const> <Constant_Def> <sem> { <const> <Constant_Def> <sem> }

<Variable_Decl> ::= <variable> <Variable_Def> <sem> { <variable> <Variable_Def> <sem> }

<Func_Def_Val> ::= <Decl_Head> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

<Func_Def_Non> ::= <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

<Main_Func> ::= <main> <lp> <rp> <lbe> <Compound_Stat> <rbe>

<Constant_Def> ::= <int> <id> <as> <Sgn_Int> { <com> <id> <as> <Sgn_Int> }
| <float> <id> <as> <Sgn_Flt> { <com> <id> <as> <Sgn_Flt> }
| <char> <id> <as> <const_ch> { <com> <id> <as> <const_ch> }
| <string> <id> <as> <const_str> { <com> <id> <as> <const_str> }

<Variable_Def> ::= <Type_id> (<id> | <id> <lbk> <const_int> <rbk>) { <com> (<id> | <id> <lbk> <const_int> <rbk>) }

<Decl_Head> ::= <Type_id> <id>

<Param> ::= <Type_id> <id> { <com> <Type_id> <id> }
| <eps>

<Compound_Stat> ::= [<Constant_Decl>] [<Variable_Decl>] <Stats>

<Stats> ::= { <Stat> }

<Stat> ::= <Cond_Stat>

| <Loop_Stat>

| <lbe> <Stats> <rbe>

| <Func_Call_Stat_Val> <sem>

| <Func_Call_Stat_Non> <sem>

| <Assign_Stat> <sem>

| <Scanf_Stat> <sem>

| <Print_Stat> <sem>

| <Ret_Stat> <sem>

| <sem>

<Cond_Stat> ::= <if> <lp> <Cond> <rp> <Stat>

<Loop_Stat> ::= <while> <lp> <Cond> <rp> <Stat>

<Func_Call_Stat_Val> ::= <id> <lp> <Value_Params> <rp>

<Func_Call_Stat_Non> ::= <id> <lp> <Value_Params> <rp>

<Assign_Stat> ::= <id> <as> <Expr>

| <id> <lbk> <Expr> <rbk> <as> <Expr>

<Scanf_Stat> ::= <scanf> <lp> <id> { <com> <id> } <rp>

<Print_Stat> ::= <printf> <lp> <const_str> <com> <Expr> <rp>

| <printf> <lp> <const_str> <rp>

| <printf> <lp> <Expr> <rp>

<Ret_Stat> ::= <return> [<Expr>]

<Cond> ::= <Expr> [<relational_op> <Expr>]

<Value_Params> ::= <Expr> { <com> <Expr> }

| <eps>

<Expr> ::= [<add_sub>] <Item> { <add_sub> <Item> }

<Item> ::= <Factor> { <mul_div> <Factor> }

<Factor> ::= <id>

| <id> <lbk> <Expr> <rbk>

| <lp> <Expr> <rp>

| <Sgn_Int>

| <Sgn_Flt>

| <Func_Call_Stat_Val>

<Sgn_Int> ::= [<add_sub>] <const_int>

<Sgn_Flt> ::= [<add_sub>] <const_flt>

<Type_id> ::= <int>

| <float>

| <char>

| <string>

附录C

根据公式消除 { } 和 [ ] 的结果如下：

<Start> ::= (<Constant_Decl> | <eps>) (<Variable_Decl> | <eps>) <Start__X_1> <Main_Func>

<Start__X_1> ::= (<Func_Def_Val> | <Func_Def_Non>) <Start__X_1>

| <eps>

<Constant_Decl> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1>

<Constant_Decl__X_1> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1>

| <eps>

<Variable_Decl> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1>

<Variable_Decl__X_1> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1>

| <eps>

<Func_Def_Val> ::= <Decl_Head> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

<Func_Def_Non> ::= <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

<Main_Func> ::= <main> <lp> <rp> <lbe> <Compound_Stat> <rbe>

<Constant_Def> ::= <int> <id> <as> <Sgn_Int> <Constant_Def__X_1>

| <float> <id> <as> <Sgn_Flt> <Constant_Def__X_2>

| <char> <id> <as> <const_ch> <Constant_Def__X_3>

| <string> <id> <as> <const_str> <Constant_Def__X_4>

<Constant_Def__X_1> ::= <com> <id> <as> <Sgn_Int> <Constant_Def__X_1>

| <eps>

<Constant_Def__X_2> ::= <com> <id> <as> <Sgn_Flt> <Constant_Def__X_2>

| <eps>

<Constant_Def__X_3> ::= <com> <id> <as> <const_ch> <Constant_Def__X_3>

| <eps>

<Constant_Def__X_4> ::= <com> <id> <as> <const_str> <Constant_Def__X_4>

| <eps>

<Variable_Def> ::= <Type_id> (<id> | <id> <lbk> <const_int> <rbk>) <Variable_Def__X_1>

<Variable_Def__X_1> ::= <com> (<id> | <id> <lbk> <const_int> <rbk>) <Variable_Def__X_1>

| <eps>

<Decl_Head> ::= <Type_id> <id>

<Param> ::= <Type_id> <id> <Param__X_1>

| <eps>

<Param__X_1> ::= <com> <Type_id> <id> <Param__X_1>

| <eps>

<Compound_Stat> ::= (<Constant_Decl> | <eps>) (<Variable_Decl> | <eps>) <Stats>

<Stats> ::= <Stats__X_1>

<Stats__X_1> ::= <Stat> <Stats__X_1>

| <eps>

<Stat> ::= <Cond_Stat>

| <Loop_Stat>

| <lbe> <Stats> <rbe>

| <Func_Call_Stat_Val> <sem>

| <Func_Call_Stat_Non> <sem>

| <Assign_Stat> <sem>

| <Scanf_Stat> <sem>

| <Print_Stat> <sem>

| <Ret_Stat> <sem>

| <sem>

<Cond_Stat> ::= <if> <lp> <Cond> <rp> <Stat>

<Loop_Stat> ::= <while> <lp> <Cond> <rp> <Stat>

<Func_Call_Stat_Val> ::= <id> <lp> <Value_Params> <rp>

<Func_Call_Stat_Non> ::= <id> <lp> <Value_Params> <rp>

<Assign_Stat> ::= <id> <as> <Expr>

| <id> <lbk> <Expr> <rbk> <as> <Expr>

<Scanf_Stat> ::= <scanf> <lp> <id> <Scanf_Stat__X_1> <rp>

<Scanf_Stat__X_1> ::= <com> <id> <Scanf_Stat__X_1>

| <eps>

<Print_Stat> ::= <printf> <lp> <const_str> <com> <Expr> <rp>

| <printf> <lp> <const_str> <rp>

| <printf> <lp> <Expr> <rp>

<Ret_Stat> ::= <return> (<Expr> | <eps>)

<Cond> ::= <Expr> (<relational_op> <Expr> | <eps>)

<Value_Params> ::= <Expr> <Value_Params__X_1>

| <eps>

<Value_Params__X_1> ::= <com> <Expr> <Value_Params__X_1>

| <eps>

<Expr> ::= (<add_sub> | <eps>) <Item> <Expr__X_1>

<Expr__X_1> ::= <add_sub> <Item> <Expr__X_1>

| <eps>

<Item> ::= <Factor> <Item__X_1>

<Item__X_1> ::= <mul_div> <Factor> <Item__X_1>

| <eps>

<Factor> ::= <id>

| <id> <lbk> <Expr> <rbk>

| <lp> <Expr> <rp>

| <Sgn_Int>

| <Sgn_Flt>

| <Func_Call_Stat_Val>

<Sgn_Int> ::= (<add_sub> | <eps>) <const_int>

<Sgn_Flt> ::= (<add_sub> | <eps>) <const_flt>

<Type_id> ::= <int>

| <float>

| <char>

| <string>

附录D

文法 G 的化简结果如下:

<Start> ::= <Constant_Decl> <Variable_Decl> <Start__X_1> <Main_Func>

| <Constant_Decl> <Start__X_1> <Main_Func>

| <Variable_Decl> <Start__X_1> <Main_Func>

| <Start__X_1> <Main_Func>

<Start__X_1> ::= <Func_Def_Val> <Start__X_1>

| <Func_Def_Non> <Start__X_1>

| <eps>

<Constant_Decl> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1>

<Constant_Decl__X_1> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1>

| <eps>

<Variable_Decl> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1>

<Variable_Decl__X_1> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1>

| <eps>

<Func_Def_Val> ::= <Decl_Head> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

<Func_Def_Non> ::= <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

<Main_Func> ::= <main> <lp> <rp> <lbe> <Compound_Stat> <rbe>

<Constant_Def> ::= <int> <id> <as> <Sgn_Int> <Constant_Def__X_1>

| <float> <id> <as> <Sgn_Flt> <Constant_Def__X_2>

| <char> <id> <as> <const_ch> <Constant_Def__X_3>

| <string> <id> <as> <const_str> <Constant_Def__X_4>

<Constant_Def__X_1> ::= <com> <id> <as> <Sgn_Int> <Constant_Def__X_1>

| <eps>

<Constant_Def__X_2> ::= <com> <id> <as> <Sgn_Flt> <Constant_Def__X_2>

| <eps>

<Constant_Def__X_3> ::= <com> <id> <as> <const_ch> <Constant_Def__X_3>

| <eps>

<Constant_Def__X_4> ::= <com> <id> <as> <const_str> <Constant_Def__X_4>

| <eps>

<Variable_Def> ::= <Type_id> <id> <Variable_Def__X_1>

| <Type_id> <id> <lbk> <const_int> <rbk> <Variable_Def__X_1>

<Variable_Def__X_1> ::= <com> <id> <Variable_Def__X_1>

| <com> <id> <lbk> <const_int> <rbk> <Variable_Def__X_1>

| <eps>

<Decl_Head> ::= <Type_id> <id>

<Param> ::= <Type_id> <id> <Param__X_1>

| <eps>

<Param__X_1> ::= <com> <Type_id> <id> <Param__X_1>

| <eps>

<Compound_Stat> ::= <Constant_Decl> <Variable_Decl> <Stats>

| <Constant_Decl> <Stats>

| <Variable_Decl> <Stats>

| <Stats>

<Stats> ::= <Stats__X_1>

<Stats__X_1> ::= <Stat> <Stats__X_1>

| <eps>

<Stat> ::= <Cond_Stat>

| <Loop_Stat>

| <lbe> <Stats> <rbe>

| <Func_Call_Stat_Val> <sem>

| <Func_Call_Stat_Non> <sem>

| <Assign_Stat> <sem>

| <Scanf_Stat> <sem>

| <Print_Stat> <sem>

| <Ret_Stat> <sem>

| <sem>

<Cond_Stat> ::= <if> <lp> <Cond> <rp> <Stat>

<Loop_Stat> ::= <while> <lp> <Cond> <rp> <Stat>

<Func_Call_Stat_Val> ::= <id> <lp> <Value_Params> <rp>

<Func_Call_Stat_Non> ::= <id> <lp> <Value_Params> <rp>

<Assign_Stat> ::= <id> <as> <Expr>

| <id> <lbk> <Expr> <rbk> <as> <Expr>

<Scanf_Stat> ::= <scanf> <lp> <id> <Scanf_Stat__X_1> <rp>

<Scanf_Stat__X_1> ::= <com> <id> <Scanf_Stat__X_1>

| <eps>

<Print_Stat> ::= <printf> <lp> <const_str> <com> <Expr> <rp>

| <printf> <lp> <const_str> <rp>

| <printf> <lp> <Expr> <rp>

<Ret_Stat> ::= <return> <Expr>

| <return>

<Cond> ::= <Expr> <relational_op> <Expr>

| <Expr>

<Value_Params> ::= <Expr> <Value_Params__X_1>

| <eps>

<Value_Params__X_1> ::= <com> <Expr> <Value_Params__X_1>

| <eps>

<Expr> ::= <add_sub> <Item> <Expr__X_1>

| <Item> <Expr__X_1>

<Expr__X_1> ::= <add_sub> <Item> <Expr__X_1>

| <eps>

<Item> ::= <Factor> <Item__X_1>

<Item__X_1> ::= <mul_div> <Factor> <Item__X_1>

| <eps>

<Factor> ::= <id>

| <id> <lbk> <Expr> <rbk>

| <lp> <Expr> <rp>

| <Sgn_Int>

| <Sgn_Flt>

| <Func_Call_Stat_Val>

<Sgn_Int> ::= <add_sub> <const_int>

| <const_int>

<Sgn_Flt> ::= <add_sub> <const_flt>

| <const_flt>

<Type_id> ::= <int>

| <float>

| <char>

| <string>

附录E

提取公共左因子的结果如下：

<Start> ::= <Constant_Decl> <Start__Y_1>

| <Variable_Decl> <Start__X_1> <Main_Func>

| <Start__X_1> <Main_Func>

<Start__Y_1> ::= <Variable_Decl> <Start__X_1> <Main_Func>

| <Start__X_1> <Main_Func>

<Start__X_1> ::= <Func_Def_Val> <Start__X_1>

| <Func_Def_Non> <Start__X_1>

| <eps>

<Constant_Decl> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1>

<Constant_Decl__X_1> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1>

| <eps>

<Variable_Decl> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1>

<Variable_Decl__X_1> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1>

| <eps>

<Func_Def_Val> ::= <Decl_Head> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

<Func_Def_Non> ::= <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

<Main_Func> ::= <main> <lp> <rp> <lbe> <Compound_Stat> <rbe>

<Constant_Def> ::= <int> <id> <as> <Sgn_Int> <Constant_Def__X_1>

| <float> <id> <as> <Sgn_Flt> <Constant_Def__X_2>

| <char> <id> <as> <const_ch> <Constant_Def__X_3>

| <string> <id> <as> <const_str> <Constant_Def__X_4>

<Constant_Def__X_1> ::= <com> <id> <as> <Sgn_Int> <Constant_Def__X_1>

| <eps>

<Constant_Def__X_2> ::= <com> <id> <as> <Sgn_Flt> <Constant_Def__X_2>

| <eps>

<Constant_Def__X_3> ::= <com> <id> <as> <const_ch> <Constant_Def__X_3>

| <eps>

<Constant_Def__X_4> ::= <com> <id> <as> <const_str> <Constant_Def__X_4>

| <eps>

<Variable_Def> ::= <Type_id> <id> <Variable_Def__Y_1>

<Variable_Def__Y_1> ::= <Variable_Def__X_1>

| <lbk> <const_int> <rbk> <Variable_Def__X_1>

<Variable_Def__X_1> ::= <com> <id> <Variable_Def__X_1__Y_1>

| <eps>

<Variable_Def__X_1__Y_1> ::= <Variable_Def__X_1>

| <lbk> <const_int> <rbk> <Variable_Def__X_1>

<Decl_Head> ::= <Type_id> <id>

<Param> ::= <Type_id> <id> <Param__X_1>

| <eps>

<Param__X_1> ::= <com> <Type_id> <id> <Param__X_1>

| <eps>

<Compound_Stat> ::= <Constant_Decl> <Compound_Stat__Y_1>

| <Variable_Decl> <Stats>

| <Stats>

<Compound_Stat__Y_1> ::= <Variable_Decl> <Stats>

| <Stats>

<Stats> ::= <Stats__X_1>

<Stats__X_1> ::= <Stat> <Stats__X_1>

| <eps>

<Stat> ::= <Cond_Stat>

| <Loop_Stat>

| <lbe> <Stats> <rbe>

| <Func_Call_Stat_Val> <sem>

| <Func_Call_Stat_Non> <sem>

| <Assign_Stat> <sem>

| <Scanf_Stat> <sem>

| <Print_Stat> <sem>

| <Ret_Stat> <sem>

| <sem>

<Cond_Stat> ::= <if> <lp> <Cond> <rp> <Stat>

<Loop_Stat> ::= <while> <lp> <Cond> <rp> <Stat>

<Func_Call_Stat_Val> ::= <id> <lp> <Value_Params> <rp>

<Func_Call_Stat_Non> ::= <id> <lp> <Value_Params> <rp>

<Assign_Stat> ::= <id> <Assign_Stat__Y_1>

<Assign_Stat__Y_1> ::= <as> <Expr>

| <lbk> <Expr> <rbk> <as> <Expr>

<Scanf_Stat> ::= <scanf> <lp> <id> <Scanf_Stat__X_1> <rp>

<Scanf_Stat__X_1> ::= <com> <id> <Scanf_Stat__X_1>

| <eps>

<Print_Stat> ::= <printf> <lp> <Print_Stat__Y_1>

<Print_Stat__Y_1> ::= <const_str> <Print_Stat__Y_1__Y_1>

| <Expr> <rp>

<Print_Stat__Y_1__Y_1> ::= <com> <Expr> <rp>

| <rp>

<Ret_Stat> ::= <return> <Ret_Stat__Y_1>

<Ret_Stat__Y_1> ::= <Expr>

| <eps>

<Cond> ::= <Expr> <Cond__Y_1>

<Cond__Y_1> ::= <relational_op> <Expr>

| <eps>

<Value_Params> ::= <Expr> <Value_Params__X_1>

| <eps>

<Value_Params__X_1> ::= <com> <Expr> <Value_Params__X_1>

| <eps>

<Expr> ::= <add_sub> <Item> <Expr__X_1>

| <Item> <Expr__X_1>

<Expr__X_1> ::= <add_sub> <Item> <Expr__X_1>

| <eps>

<Item> ::= <Factor> <Item__X_1>

<Item__X_1> ::= <mul_div> <Factor> <Item__X_1>

| <eps>

<Factor> ::= <id> <Factor__Y_1>

| <lp> <Expr> <rp>

| <Sgn_Int>

| <Sgn_Flt>

| <Func_Call_Stat_Val>

<Factor__Y_1> ::= <lbk> <Expr> <rbk>

| <eps>

<Sgn_Int> ::= <add_sub> <const_int>

| <const_int>

<Sgn_Flt> ::= <add_sub> <const_flt>

| <const_flt>

<Type_id> ::= <int>

| <float>

| <char>

| <string>

附录F

消除冗余后的结果如下：

<Start> ::= <Constant_Decl> <Start__Y_1>

| <Variable_Decl> <Start__X_1> <Main_Func>

| <Start__X_1> <Main_Func>

<Start__Y_1> ::= <Variable_Decl> <Start__X_1> <Main_Func>

| <Start__X_1> <Main_Func>

<Start__X_1> ::= <Func_Def_Val> <Start__X_1>

| <Func_Def_Non> <Start__X_1>

| <eps>

<Constant_Decl> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1>

<Constant_Decl__X_1> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1>

| <eps>

<Variable_Decl> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1>

<Variable_Decl__X_1> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1>

| <eps>

<Func_Def_Val> ::= <Decl_Head> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

<Func_Def_Non> ::= <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

<Main_Func> ::= <main> <lp> <rp> <lbe> <Compound_Stat> <rbe>

<Constant_Def> ::= <int> <id> <as> <Sgn_Int> <Constant_Def__X_1>

| <float> <id> <as> <Sgn_Flt> <Constant_Def__X_2>

| <char> <id> <as> <const_ch> <Constant_Def__X_3>

| <string> <id> <as> <const_str> <Constant_Def__X_4>

<Constant_Def__X_1> ::= <com> <id> <as> <Sgn_Int> <Constant_Def__X_1>

| <eps>

<Constant_Def__X_2> ::= <com> <id> <as> <Sgn_Flt> <Constant_Def__X_2>

| <eps>

<Constant_Def__X_3> ::= <com> <id> <as> <const_ch> <Constant_Def__X_3>

| <eps>

<Constant_Def__X_4> ::= <com> <id> <as> <const_str> <Constant_Def__X_4>

| <eps>

<Variable_Def> ::= <Type_id> <id> <Variable_Def__Y_1>

<Variable_Def__Y_1> ::= <Variable_Def__X_1>

| <lbk> <const_int> <rbk> <Variable_Def__X_1>

<Variable_Def__X_1> ::= <com> <id> <Variable_Def__Y_1>

| <eps>

<Decl_Head> ::= <Type_id> <id>

<Param> ::= <Type_id> <id> <Param__X_1>

| <eps>

<Param__X_1> ::= <com> <Type_id> <id> <Param__X_1>

| <eps>

<Compound_Stat> ::= <Constant_Decl> <Compound_Stat__Y_1>

| <Variable_Decl> <Stats>

| <Stats>

<Compound_Stat__Y_1> ::= <Variable_Decl> <Stats>

| <Stats>

<Stats> ::= <Stat> <Stats>

| <eps>

<Stat> ::= <Cond_Stat>

| <Loop_Stat>

| <lbe> <Stats> <rbe>

| <Func_Call_Stat_Val> <sem>

| <Assign_Stat> <sem>

| <Scanf_Stat> <sem>

| <Print_Stat> <sem>

| <Ret_Stat> <sem>

| <sem>

<Cond_Stat> ::= <if> <lp> <Cond> <rp> <Stat>

<Loop_Stat> ::= <while> <lp> <Cond> <rp> <Stat>

<Func_Call_Stat_Val> ::= <id> <lp> <Value_Params> <rp>

<Assign_Stat> ::= <id> <Assign_Stat__Y_1>

<Assign_Stat__Y_1> ::= <as> <Expr>

| <lbk> <Expr> <rbk> <as> <Expr>

<Scanf_Stat> ::= <scanf> <lp> <id> <Scanf_Stat__X_1> <rp>

<Scanf_Stat__X_1> ::= <com> <id> <Scanf_Stat__X_1>

| <eps>

<Print_Stat> ::= <printf> <lp> <Print_Stat__Y_1>

<Print_Stat__Y_1> ::= <const_str> <Print_Stat__Y_1__Y_1>

| <Expr> <rp>

<Print_Stat__Y_1__Y_1> ::= <com> <Expr> <rp>

| <rp>

<Ret_Stat> ::= <return> <Ret_Stat__Y_1>

<Ret_Stat__Y_1> ::= <Expr>

| <eps>

<Cond> ::= <Expr> <Cond__Y_1>

<Cond__Y_1> ::= <relational_op> <Expr>

| <eps>

<Value_Params> ::= <Expr> <Value_Params__X_1>

| <eps>

<Value_Params__X_1> ::= <com> <Expr> <Value_Params__X_1>

| <eps>

<Expr> ::= <add_sub> <Item> <Expr__X_1>

| <Item> <Expr__X_1>

<Expr__X_1> ::= <add_sub> <Item> <Expr__X_1>

| <eps>

<Item> ::= <Factor> <Item__X_1>

<Item__X_1> ::= <mul_div> <Factor> <Item__X_1>

| <eps>

<Factor> ::= <id> <Factor__Y_1>

| <lp> <Expr> <rp>

| <Sgn_Int>

| <Sgn_Flt>

| <Func_Call_Stat_Val>

<Factor__Y_1> ::= <lbk> <Expr> <rbk>

| <eps>

<Sgn_Int> ::= <add_sub> <const_int>

| <const_int>

<Sgn_Flt> ::= <add_sub> <const_flt>

| <const_flt>

<Type_id> ::= <int>

| <float>

| <char>

| <string>

消除间接左递归的结果如下：

<Start> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1> <Constant_Decl> <Start__Y_1>
| <variable> <Variable_Def> <sem> <Variable_Decl__X_1> <Start__X_1> <Main_Func>
| <int> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>
| <float> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>
| <char> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>
| <string> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>
| <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>
| <main> <lp> <rp> <lbe> <Compound_Stat> <rbe>

<Start__Y_1> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1> <Start__X_1> <Main_Func>
| <int> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>
| <float> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>
| <char> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>
| <string> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>
| <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>
| <main> <lp> <rp> <lbe> <Compound_Stat> <rbe>

<Start__X_1> ::= <int> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>
| <float> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>
| <char> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>
| <string> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>
| <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>
| <eps>

<Constant_Decl__X_1> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1>

| <eps>

<Variable_Decl__X_1> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1>

| <eps>

<Func_Def_Val> ::= <int> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

| <float> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

| <char> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

| <string> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

<Func_Def_Non> ::= <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe>

<Main_Func> ::= <main> <lp> <rp> <lbe> <Compound_Stat> <rbe>

<Constant_Def> ::= <int> <id> <as> <Sgn_Int> <Constant_Def__X_1>

| <float> <id> <as> <Sgn_Flt> <Constant_Def__X_2>

| <char> <id> <as> <const_ch> <Constant_Def__X_3>

| <string> <id> <as> <const_str> <Constant_Def__X_4>

<Constant_Def__X_1> ::= <com> <id> <as> <Sgn_Int> <Constant_Def__X_1>

| <eps>

<Constant_Def__X_2> ::= <com> <id> <as> <Sgn_Flt> <Constant_Def__X_2>

| <eps>

<Constant_Def__X_3> ::= <com> <id> <as> <const_ch> <Constant_Def__X_3>

| <eps>

<Constant_Def__X_4> ::= <com> <id> <as> <const_str> <Constant_Def__X_4>

| <eps>

<Variable_Def> ::= <int> <id> <Variable_Def__Y_1>

| <float> <id> <Variable_Def__Y_1>

| <char> <id> <Variable_Def__Y_1>

| <string> <id> <Variable_Def__Y_1>

<Variable_Def__Y_1> ::= <com> <id> <Variable_Def__Y_1>

| <lbk> <const_int> <rbk> <Variable_Def__X_1>

| <eps>

<Variable_Def__X_1> ::= <com> <id> <Variable_Def__Y_1>

| <eps>

<Decl_Head> ::= <int> <id>

| <float> <id>

| <char> <id>

| <string> <id>

<Param> ::= <int> <id> <Param__X_1>

| <float> <id> <Param__X_1>

| <char> <id> <Param__X_1>

| <string> <id> <Param__X_1>

| <eps>

<Param__X_1> ::= <com> <Type_id> <id> <Param__X_1>

| <eps>

<Compound_Stat> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1> <Compound_Sta t__Y_1>

| <variable> <Variable_Def> <sem> <Variable_Decl__X_1> <Stats>

| <if> <lp> <Cond> <rp> <Stat> <Stats>

| <while> <lp> <Cond> <rp> <Stat> <Stats>

| <lbe> <Stats> <rbe> <Stats>

| <id> <Stat__Y_1> <Stats>

| <scanf> <lp> <id> <Scanf_Stat__X_1> <rp> <sem> <Stats>

| <printf> <lp> <Print_Stat__Y_1> <sem> <Stats>

| <return> <Ret_Stat__Y_1> <sem> <Stats>

| <sem> <Stats>

| <eps>

<Compound_Stat__Y_1> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1> <Stats>

| <if> <lp> <Cond> <rp> <Stat> <Stats>

| <while> <lp> <Cond> <rp> <Stat> <Stats>

| <lbe> <Stats> <rbe> <Stats>

| <id> <Stat__Y_1> <Stats>

| <scanf> <lp> <id> <Scanf_Stat__X_1> <rp> <sem> <Stats>

| <printf> <lp> <Print_Stat__Y_1> <sem> <Stats>

| <return> <Ret_Stat__Y_1> <sem> <Stats>

| <sem> <Stats>

| <eps>

<Constant_Decl> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1>

<Variable_Decl> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1>

<Stats> ::= <if> <lp> <Cond> <rp> <Stat> <Stats>

| <while> <lp> <Cond> <rp> <Stat> <Stats>

| <lbe> <Stats> <rbe> <Stats>

| <id> <Stat__Y_1> <Stats>

| <scanf> <lp> <id> <Scanf_Stat__X_1> <rp> <sem> <Stats>

| <printf> <lp> <Print_Stat__Y_1> <sem> <Stats>

| <return> <Ret_Stat__Y_1> <sem> <Stats>

| <sem> <Stats>

| <eps>

<Stat> ::= <if> <lp> <Cond> <rp> <Stat>

| <while> <lp> <Cond> <rp> <Stat>

| <lbe> <Stats> <rbe>

| <id> <Stat__Y_1>

| <scanf> <lp> <id> <Scanf_Stat__X_1> <rp> <sem>

| <printf> <lp> <Print_Stat__Y_1> <sem>

| <return> <Ret_Stat__Y_1> <sem>

| <sem>

<Stat__Y_1> ::= <lp> <Value_Params> <rp> <sem>

| <as> <Expr> <sem>

| <lbk> <Expr> <rbk> <as> <Expr> <sem>

<Cond_Stat> ::= <if> <lp> <Cond> <rp> <Stat>

<Loop_Stat> ::= <while> <lp> <Cond> <rp> <Stat>

<Assign_Stat> ::= <id> <Assign_Stat__Y_1>

<Assign_Stat__Y_1> ::= <as> <Expr>

| <lbk> <Expr> <rbk> <as> <Expr>

<Scanf_Stat> ::= <scanf> <lp> <id> <Scanf_Stat__X_1> <rp>

<Scanf_Stat__X_1> ::= <com> <id> <Scanf_Stat__X_1>

| <eps>

<Print_Stat> ::= <printf> <lp> <Print_Stat__Y_1>

<Print_Stat__Y_1> ::= <const_str> <Print_Stat__Y_1__Y_1>

| <add_sub> <Expr__Y_1> <rp>

| <id> <Factor__Y_1> <Item__X_1> <Expr__X_1> <rp>

| <lp> <Expr> <rp> <Item__X_1> <Expr__X_1> <rp>

| <const_int> <Item__X_1> <Expr__X_1> <rp>

| <const_flt> <Item__X_1> <Expr__X_1> <rp>

<Print_Stat__Y_1__Y_1> ::= <com> <Expr> <rp>

| <rp>

<Ret_Stat> ::= <return> <Ret_Stat__Y_1>

<Ret_Stat__Y_1> ::= <add_sub> <Expr__Y_1>

| <id> <Factor__Y_1> <Item__X_1> <Expr__X_1>

| <lp> <Expr> <rp> <Item__X_1> <Expr__X_1>

| <const_int> <Item__X_1> <Expr__X_1>

| <const_flt> <Item__X_1> <Expr__X_1>

| <eps>

<Cond> ::= <add_sub> <Expr__Y_1> <Cond__Y_1>

| <id> <Factor__Y_1> <Item__X_1> <Expr__X_1> <Cond__Y_1>

| <lp> <Expr> <rp> <Item__X_1> <Expr__X_1> <Cond__Y_1>

| <const_int> <Item__X_1> <Expr__X_1> <Cond__Y_1>

| <const_flt> <Item__X_1> <Expr__X_1> <Cond__Y_1>

<Cond__Y_1> ::= <relational_op> <Expr>

| <eps>

<Value_Params> ::= <add_sub> <Expr__Y_1> <Value_Params__X_1>

| <id> <Factor__Y_1> <Item__X_1> <Expr__X_1> <Value_Params__X_1>

| <lp> <Expr> <rp> <Item__X_1> <Expr__X_1> <Value_Params__X_1>

| <const_int> <Item__X_1> <Expr__X_1> <Value_Params__X_1>

| <const_flt> <Item__X_1> <Expr__X_1> <Value_Params__X_1>

| <eps>

<Value_Params__X_1> ::= <com> <Expr> <Value_Params__X_1>

| <eps>

<Expr> ::= <add_sub> <Expr__Y_1>

| <id> <Factor__Y_1> <Item__X_1> <Expr__X_1>

| <lp> <Expr> <rp> <Item__X_1> <Expr__X_1>

| <const_int> <Item__X_1> <Expr__X_1>

| <const_flt> <Item__X_1> <Expr__X_1>

<Expr__Y_1> ::= <id> <Factor__Y_1> <Item__X_1> <Expr__X_1>

| <lp> <Expr> <rp> <Item__X_1> <Expr__X_1>

| <add_sub> <Factor__Y_2> <Item__X_1> <Expr__X_1>

| <const_int> <Item__X_1> <Expr__X_1>

| <const_flt> <Item__X_1> <Expr__X_1>

<Expr__X_1> ::= <add_sub> <Item> <Expr__X_1>

| <eps>

<Item> ::= <id> <Factor__Y_1> <Item__X_1>

| <lp> <Expr> <rp> <Item__X_1>

| <add_sub> <Factor__Y_2> <Item__X_1>

| <const_int> <Item__X_1>

| <const_flt> <Item__X_1>

<Item__X_1> ::= <mul_div> <Factor> <Item__X_1>

| <eps>

<Factor> ::= <id> <Factor__Y_1>

| <lp> <Expr> <rp>

| <add_sub> <Factor__Y_2>

| <const_int>

| <const_flt>

<Factor__Y_1> ::= <lbk> <Expr> <rbk>

| <lp> <Value_Params> <rp>

| <eps>

<Factor__Y_2> ::= <const_int>

| <const_flt>

<Sgn_Int> ::= <add_sub> <const_int>

| <const_int>

<Sgn_Flt> ::= <add_sub> <const_flt>

| <const_flt>

<Type_id> ::= <int>

| <float>

| <char>

| <string>

<Func_Call_Stat_Val> ::= <id> <lp> <Value_Params> <rp>

附录H

消除关于文法G中多余的产生式的结果如下：

<Start> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1> <Start__Y_1>

| <variable> <Variable_Def> <sem> <Variable_Decl__X_1> <Start__X_1> <Main_Func>

| <int> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>

| <float> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>

| <char> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>

| <string> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>

| <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>

| <main> <lp> <rp> <lbe> <Compound_Stat> <rbe>

<Start__Y_1> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1> <Start__X_1> <Main_Func>

| <int> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>

| <float> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>

| <char> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>

| <string> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>

| <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1> <Main_Func>

| <main> <lp> <rp> <lbe> <Compound_Stat> <rbe>

<Start__X_1> ::= <int> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>

| <float> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>

| <char> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>

| <string> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>

| <void> <id> <lp> <Param> <rp> <lbe> <Compound_Stat> <rbe> <Start__X_1>

| <eps>

<Constant_Decl__X_1> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1>

| <eps>

<Variable_Decl__X_1> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1>

| <eps>

<Main_Func> ::= <main> <lp> <rp> <lbe> <Compound_Stat> <rbe>

<Constant_Def> ::= <int> <id> <as> <Sgn_Int> <Constant_Def__X_1>

| <float> <id> <as> <Sgn_Flt> <Constant_Def__X_2>

| <char> <id> <as> <const_ch> <Constant_Def__X_3>

| <string> <id> <as> <const_str> <Constant_Def__X_4>

<Constant_Def__X_1> ::= <com> <id> <as> <Sgn_Int> <Constant_Def__X_1>

| <eps>

<Constant_Def__X_2> ::= <com> <id> <as> <Sgn_Flt> <Constant_Def__X_2>

| <eps>

<Constant_Def__X_3> ::= <com> <id> <as> <const_ch> <Constant_Def__X_3>

| <eps>

<Constant_Def__X_4> ::= <com> <id> <as> <const_str> <Constant_Def__X_4>

| <eps>

<Variable_Def> ::= <int> <id> <Variable_Def__Y_1>

| <float> <id> <Variable_Def__Y_1>

| <char> <id> <Variable_Def__Y_1>

| <string> <id> <Variable_Def__Y_1>

<Variable_Def__X_1> ::= <com> <id> <Variable_Def__Y_1>

| <eps>

<Variable_Def__Y_1> ::= <com> <id> <Variable_Def__Y_1>

| <lbk> <const_int> <rbk> <Variable_Def__X_1>

| <eps>

<Param> ::= <int> <id> <Param__X_1>

| <float> <id> <Param__X_1>

| <char> <id> <Param__X_1>

| <string> <id> <Param__X_1>

| <eps>

<Param__X_1> ::= <com> <Type_id> <id> <Param__X_1>

| <eps>

<Compound_Stat> ::= <const> <Constant_Def> <sem> <Constant_Decl__X_1> <Compound_Sta
t__Y_1>

| <variable> <Variable_Def> <sem> <Variable_Decl__X_1> <Stats>

| <if> <lp> <Cond> <rp> <Stat> <Stats>

| <while> <lp> <Cond> <rp> <Stat> <Stats>

| <lbe> <Stats> <rbe> <Stats>

| <id> <Stat__Y_1> <Stats>

| <scanf> <lp> <id> <Scanf_Stat__X_1> <rp> <sem> <Stats>

| <printf> <lp> <Print_Stat__Y_1> <sem> <Stats>

| <return> <Ret_Stat__Y_1> <sem> <Stats>

| <sem> <Stats>

| <eps>

<Compound_Stat__Y_1> ::= <variable> <Variable_Def> <sem> <Variable_Decl__X_1> <Stats>

| <if> <lp> <Cond> <rp> <Stat> <Stats>

| <while> <lp> <Cond> <rp> <Stat> <Stats>

| <lbe> <Stats> <rbe> <Stats>

| <id> <Stat__Y_1> <Stats>

| <scanf> <lp> <id> <Scanf_Stat__X_1> <rp> <sem> <Stats>

| <printf> <lp> <Print_Stat__Y_1> <sem> <Stats>

| <return> <Ret_Stat__Y_1> <sem> <Stats>

| <sem> <Stats>

| <eps>

<Stats> ::= <if> <lp> <Cond> <rp> <Stat> <Stats>

| <while> <lp> <Cond> <rp> <Stat> <Stats>

| <lbe> <Stats> <rbe> <Stats>

| <id> <Stat__Y_1> <Stats>

| <scanf> <lp> <id> <Scanf_Stat__X_1> <rp> <sem> <Stats>

| <printf> <lp> <Print_Stat__Y_1> <sem> <Stats>

| <return> <Ret_Stat__Y_1> <sem> <Stats>

| <sem> <Stats>

| <eps>

<Stat> ::= <if> <lp> <Cond> <rp> <Stat>

| <while> <lp> <Cond> <rp> <Stat>

| <lbe> <Stats> <rbe>

| <id> <Stat__Y_1>

| <scanf> <lp> <id> <Scanf_Stat__X_1> <rp> <sem>

| <printf> <lp> <Print_Stat__Y_1> <sem>

| <return> <Ret_Stat__Y_1> <sem>

| <sem>

<Stat__Y_1> ::= <lp> <Value_Params> <rp> <sem>

| <as> <Expr> <sem>

| <lbk> <Expr> <rbk> <as> <Expr> <sem>

<Scanf_Stat__X_1> ::= <com> <id> <Scanf_Stat__X_1>

| <eps>

<Print_Stat__Y_1> ::= <const_str> <Print_Stat__Y_1__Y_1>

| <add_sub> <Expr__Y_1> <rp>

| <id> <Factor__Y_1> <Item__X_1> <Expr__X_1> <rp>

| <lp> <Expr> <rp> <Item__X_1> <Expr__X_1> <rp>

| <const_int> <Item__X_1> <Expr__X_1> <rp>

| <const_flt> <Item__X_1> <Expr__X_1> <rp>

<Print_Stat__Y_1__Y_1> ::= <com> <Expr> <rp>

| <rp>

<Ret_Stat__Y_1> ::= <add_sub> <Expr__Y_1>

| <id> <Factor__Y_1> <Item__X_1> <Expr__X_1>

| <lp> <Expr> <rp> <Item__X_1> <Expr__X_1>

| <const_int> <Item__X_1> <Expr__X_1>

| <const_flt> <Item__X_1> <Expr__X_1>

| <eps>

<Cond> ::= <add_sub> <Expr__Y_1> <Cond__Y_1>

| <id> <Factor__Y_1> <Item__X_1> <Expr__X_1> <Cond__Y_1>

| <lp> <Expr> <rp> <Item__X_1> <Expr__X_1> <Cond__Y_1>

| <const_int> <Item__X_1> <Expr__X_1> <Cond__Y_1>

| <const_flt> <Item__X_1> <Expr__X_1> <Cond__Y_1>

<Cond__Y_1> ::= <relational_op> <Expr>

| <eps>

<Value_Params> ::= <add_sub> <Expr__Y_1> <Value_Params__X_1>

| <id> <Factor__Y_1> <Item__X_1> <Expr__X_1> <Value_Params__X_1>

| <lp> <Expr> <rp> <Item__X_1> <Expr__X_1> <Value_Params__X_1>

| <const_int> <Item__X_1> <Expr__X_1> <Value_Params__X_1>

| <const_flt> <Item__X_1> <Expr__X_1> <Value_Params__X_1>

| <eps>

<Value_Params__X_1> ::= <com> <Expr> <Value_Params__X_1>

| <eps>

<Expr> ::= <add_sub> <Expr__Y_1>

| <id> <Factor__Y_1> <Item__X_1> <Expr__X_1>

| <lp> <Expr> <rp> <Item__X_1> <Expr__X_1>

| <const_int> <Item__X_1> <Expr__X_1>

| <const_flt> <Item__X_1> <Expr__X_1>

<Expr__Y_1> ::= <id> <Factor__Y_1> <Item__X_1> <Expr__X_1>

| <lp> <Expr> <rp> <Item__X_1> <Expr__X_1>

| <add_sub> <Factor__Y_2> <Item__X_1> <Expr__X_1>

| <const_int> <Item__X_1> <Expr__X_1>

| <const_flt> <Item__X_1> <Expr__X_1>

<Expr__X_1> ::= <add_sub> <Item> <Expr__X_1>

| <eps>

<Item> ::= <id> <Factor__Y_1> <Item__X_1>

| <lp> <Expr> <rp> <Item__X_1>

| <add_sub> <Factor__Y_2> <Item__X_1>

| <const_int> <Item__X_1>

| <const_flt> <Item__X_1>

<Item__X_1> ::= <mul_div> <Factor> <Item__X_1>

| <eps>

<Factor> ::= <id> <Factor__Y_1>

| <lp> <Expr> <rp>

| <add_sub> <Factor__Y_2>

| <const_int>

| <const_flt>

<Factor__Y_1> ::= <lbk> <Expr> <rbk>

| <lp> <Value_Params> <rp>

| <eps>

<Factor__Y_2> ::= <const_int>

| <const_flt>

<Sgn_Int> ::= <add_sub> <const_int>

| <const_int>

<Sgn_Flt> ::= <add_sub> <const_flt>

| <const_flt>

<Type_id> ::= <int>

| <float>

| <char>

| <string>