编译原理实验

实验三：词法分析、语法分析程序实验

姓名：陈家豪

班级：2018 级计科 1 班

学号：18308013

目录

1 实验目的

扩充已有的样例语言 TINY，为扩展 TINY 语言 TINY ＋构造词法分析和语法分析程序，从而掌握词法分析和语法分析程序的构造方法。

2 实验内容

了解样例语言 TINY 及 TINY 编译器的实现，了解扩展 TINY 语言 TINY ＋，用 EBNF 描述 TINY ＋的语法，用 C 语言扩展 TINY 的词法分析和语法分析程序，构造 TINY ＋的语法分析器。

3 实验要求

将 TINY ＋源程序翻译成对应的 TOKEN 序列，并能检查一定的词法错误。将 TOKEN 序列转换成语法分析树，并能检查一定的语法错误。

4 算法原理与描述

4.1 TINY 语言定义

TINY 语言是一种小型语言，结构简单，方便实现。本次实验，我们要对 TINY 语言进行扩展，并实现词法分析和语法分析。

4.1.1 TINY Lexicon

TINY 的词法如下：

(1) 关键字（Keywords）：关键字又称为保留字，具有特殊含义和功能，用户不能设置关键字作为标识符。TINY 的关键字有 **IF**，**ELSE**，**WRITE**，**READ**，**RETURN**，**BEGIN**，**END**，**MAIN**，**INT** 和 **REAL**；

(2) 单符间隔符（Single-character separators）：间隔符具有分隔语句、公式等功能。TINY 的有，;，,，(和)；

(3) 单符操作符（Single-character operators）：单符操作符由一个符号组成，有 +，-，* 和/；

(4) 多符操作符（Multi-character operators）：多符操作符由两个及以上符号组成，有:=，== 和!=；

(5) 标识符（Identifier）：标识符由一个及以上的字母或阿拉伯数字组成，只有当一个字母和阿拉伯数字组成的序列：

- 第一位是字母而不是阿拉伯数字;
- 不是关键字;
- 不在字符串和注释内;

该序列才为标识符；

(6) 数字（Number）：数字是由阿拉伯数字和至多一个小数点组成的序列。其生成规则如下：

- $Number \rightarrow Digits \mid Digits\ '.'\ Digits$
- $Digits \rightarrow Digit \mid Digit\ Digits$
- $Digit \rightarrow\ '0' \mid '1' \mid '2' \mid '3' \mid '4' \mid '5' \mid '6' \mid '7' \mid '8' \mid '9'$

(7) 注释（Comments）：注释是一个首尾分别为/** 和 **/的字符序列，长度可以为多行；

(8) 字符串（String）：字符串是一个首尾均为”、内部没有” 的字符序列；

4.1.2 TINY Grammar

我们将 TINY 语法划分为三个层次来描述，分别是 high-level program structures, statements 和 expressions。在描述过程中，我们使用斜体单词表示非终结符号；黑色加粗正体单词和用 ' 括起来的符号表示终结符号；$(example)^*$ 表示 $example$ 重复零至多次；$[example]$ 表示 $example$ 可选；$(ex_1 \mid \cdots \mid ex_n)$ 表示从 n 个符号中选择 1 个。

对于 high-level program structures，其 EBNF 描述如下：

$$
\begin{aligned}
Program &\rightarrow FunctionDecl\ \ FunctionDecl^* \\
FunctionDecl &\rightarrow Type\ \ [\ \textbf{MAIN}\ ]\ \ \textbf{id}\ \ '('\ \ [\ FormalParams\ ]\ \ ')'\ \ Block \\
FormalParams &\rightarrow FormalParam\ \ (\ ','\ \ FormalParam\ )^* \\
FormalParam &\rightarrow Type\ \ \textbf{id} \\
Type &\rightarrow \textbf{INT}\ \mid\ \textbf{REAL}
\end{aligned}
$$

对于 statements，其 EBNF 描述如下：

$$
\begin{aligned}
Block &\rightarrow \textbf{BEGIN}\ \ Statement^*\ \ \textbf{END} \\
Statement &\rightarrow Block \\
&\quad \mid LocalVarDecl \\
&\quad \mid AssignStmt \\
&\quad \mid ReturnStmt \\
&\quad \mid IfStmt \\
&\quad \mid WriteStmt \\
&\quad \mid ReadStmt \\
LocalVarDecl &\rightarrow Type\ \ \textbf{id}\ \ ';' \\
AssignStmt &\rightarrow \textbf{id}\ \ ':='\ \ Expression\ \ ';' \\
ReturnStmt &\rightarrow \textbf{RETURN}\ \ Expression\ \ ';' \\
IfStmt &\rightarrow \textbf{IF}\ \ '('\ \ BoolMultiExpr\ \ ')'\ \ Statement\ \ [\textbf{ELSE}\ \ Statement] \\
WriteStmt &\rightarrow \textbf{WRITE}\ \ '('\ \ Expression\ \ ','\ \ \textbf{string}\ \ ')'\ \ ';' \\
ReadStmt &\rightarrow \textbf{READ}\ \ '('\ \ \textbf{id}\ \ ','\ \ \textbf{string}\ \ ')'\ \ ';'
\end{aligned}
$$

对于 expressions，其 EBNF 描述如下：

$$
\begin{aligned}
Expression &\rightarrow MultiplicativeExpr\ \ (\ (\ '+'\ \mid\ '\text{-}'\ )\ \ MultiplicativeExpr\ )^{*} \\
MultiplicativeExpr &\rightarrow PrimaryExpr\ \ (\ (\ '*'\ \mid\ '/'\ )\ \ PrimaryExpr\ )^{*} \\
PrimaryExpr &\rightarrow \mathbf{num} \\
&\quad \mid\ \mathbf{id} \\
&\quad \mid\ '('\ \ Expression\ \ ')' \\
&\quad \mid\ \mathbf{id}\ \ '('\ \ [\ ActualParams\ ]\ \ ')' \\
BoolExpression &\rightarrow Expression\ \ (\ '!='\ \mid\ '=='\ )\ \ Expression \\
ActualParams &\rightarrow Expression\ \ (\ ','\ \ Expression\ )^{*}
\end{aligned}
$$

4.2 TINY+ 语言定义

TINY+ 是在 TINY 的基础上进行扩展的语言。上网查找相关资料后，我发现人们并没有一种对 TINY+ 的严格定义。因此，我选择对 TINY 自行扩展，添加一些我认为有必要的内容、形成 TINY+；

4.2.1 TINY+ Lexicon

相比于 TINY，TINY+ 增加了以下关键字：**WHILE**,**AND**,**OR**,**BOOL**,**TRUE** 和 **FALSE**。其中 **WHILE** 用于增加对 while 语句的支持，其余 5 个新增关键字用于增加对布尔值的支持。

此外，单符操作符中增加了 % 用于求余运算，以及 ! 用于否定；增加了单符比较符 > 和 <，以及多符比较符 >= 和 <=。

4.2.2 TINY+ Grammar

相比于 TINY，TINY+ 增加了以下语法支持：

(1) 支持声明定义布尔类型变量和布尔类型函数；

(2) 支持 while 语句；

(3) 允许在声明语句中同时声明多个同一类型的变量并赋值；

(4) 支持求模（%）运算；

(5) 支持 <,≤,> 和 ≥ 比较运算；

(6) 支持在判断中使用 **AND** 和 **OR**，含义类似于 C/C++ 的 && 和 ||；

(7) 支持使用布尔表达式对变量赋值；

(8) 支持单个符号加分号组成语句，如 “x;” 是允许的（虽然没有意义）；

TINY+ 的语法结构依然分为三个层次。对于 high-level program structures，其 EBNF 描述如下：

$$
\begin{aligned}
Program &\rightarrow FunctionDecl \;\; FunctionDecl^{*} \\
FunctionDecl &\rightarrow Type \;\; [\; \textbf{MAIN} \;] \;\; \textbf{id} \;\; '(' \;\; [\; FormalParams \;] \;\; ')' \;\; Block \\
FormalParams &\rightarrow FormalParam \;\; (\; ',' \;\; FormalParam \;)^{*} \\
FormalParam &\rightarrow Type \;\; \textbf{id} \\
Type &\rightarrow \textbf{INT} \;\mid\; \textbf{REAL} \;\mid\; \textbf{BOOL}
\end{aligned}
$$

相较于 TINY，*Type* 增加了与 **BOOL** 有关的修改。

对于 statements，其 EBNF 描述如下：

$$
\begin{aligned}
Block &\rightarrow \textbf{BEGIN} \;\; Statement^{*} \;\; \textbf{END} \\
Statement &\rightarrow Block \\
&\mid LocalVarDecl \\
&\mid AssignStmt \\
&\mid ReturnStmt \\
&\mid IfStmt \\
&\mid WhileStmt \\
&\mid WriteStmt \\
&\mid ReadStmt \\
LocalVarDecl &\rightarrow Type \;\; AssignExpr \;\; (\; ',' \;\; AssignExpr \;)^{*} \;\; ';' \\
AssignStmt &\rightarrow AssignExpr \;\; ';' \\
ReturnStmt &\rightarrow \textbf{RETURN} \;\; BoolMultiExpr \;\; ';' \\
IfStmt &\rightarrow \textbf{IF} \;\; '(' \;\; BoolMultiExpr \;\; ')' \;\; Statement \;\; [\textbf{ELSE} \;\; Statement] \\
WhileStmt &\rightarrow \textbf{WHILE} \;\; '(' \;\; BoolMultiExpr \;\; ')' \;\; Statement \\
WriteStmt &\rightarrow \textbf{WRITE} \;\; '(' \;\; BoolMultiExpr \;\; ',' \;\; \textbf{string} \;\; ')' \;\; ';' \\
ReadStmt &\rightarrow \textbf{READ} \;\; '(' \;\; \textbf{id} \;\; ',' \;\; \textbf{string} \;\; ')' \;\; ';'
\end{aligned}
$$

相较于 TINY，*Statement* 增加了与 *WhileStmt* 有关的修改；新增了 *WhileStmt* 并对其进行了相关描述；*LocalVarDecl* 进行了修改，支持多变量声明与赋值；*ReturnStmt* 和 *WriteStmt* 的 *Expression* 升级为 *BoolMultiExpr*。

对于 expressions，其 EBNF 描述如下：

$$
\begin{aligned}
Expression &\rightarrow MultiplicativeExpr\ \ (\ (\ '+'\ \mid\ '-'\ )\ \ MultiplicativeExpr\ )^{*} \\
MultiplicativeExpr &\rightarrow PrimaryExpr\ \ (\ (\ '*'\ \mid\ '/'\ \mid\ '\%'\ )\ \ PrimaryExpr\ )^{*} \\
PrimaryExpr &\rightarrow \mathbf{num} \\
&\quad \mid\ \mathbf{id} \\
&\quad \mid\ \mathbf{TRUE} \\
&\quad \mid\ \mathbf{FALSE} \\
&\quad \mid\ '('\ \ BoolMultiExpr\ \ ')' \\
&\quad \mid\ \mathbf{id}\ \ '('\ \ [\ ActualParams\ ]\ \ ')' \\
AssignExpr &\rightarrow \mathbf{id}\ \ [\ ':='\ \ BoolMultiExpr\ ] \\
BoolMultiExpr &\rightarrow BoolExpression\ \ (\ (\ '\mathbf{AND}'\ \mid\ '\mathbf{OR}'\ )\ \ BoolExpression\ )^{*} \\
BoolExpression &\rightarrow Expression\ \ [\ (\ '!='\ \mid\ '=='\ \mid\ '>='\ \mid\ '<='\ \mid\ '>'\ \mid\ '<'\ )\ \ Expression\ ] \\
&\quad \mid\ '!'\ \ Expression \\
ActualParams &\rightarrow Expression\ \ (\ ','\ \ Expression\ )^{*}
\end{aligned}
$$

相比于 TINY，*MultiplicativeExpr* 增加了对 % 的支持；*PrimaryExpr* 增加了对 **TRUE** 和 **FALSE** 的支持，并将 *Epression* 改为 *BoolMultiExpr*；新增 *AssignExpr* 和 *BoolMultiExpr*，并对它们进行了相关描述；*BoolExpression* 增加了对新增比较符的支持。

以上就是 TINY+ 语法的 EBNF 描述。

4.3 词法分析器

词法分析器负责将 TINY+ 源程序翻译成对应的 TOKEN 序列，并检查词法错误。本次实验，我主要使用 DFA 识别 TOKEN。程序初始化 DFA 后开始读取字符，每读取一个字符，DFA 跳转到对应状态，直到跳转到结束状态后，返回对应的识别结果和 TOKEN。然后，再次初始化 DFA、获取下一个 TOKEN，循环操作直到源程序文本识别完毕。

本次实验我定义了以下 DFA 状态：

(1) DFA_START

(2) DFA_ID

(3) DFA_INTNUM

(4) DFA_INT2REAL

(5) DFA_REALNUM

(6) DFA_DIV_OR_COMMENT

(7) DFA_COMMENT_START

(8) DFA_COMMENT

(9) DFA_COMMENT_END

(10) DFA_COMMENT_END2

(11) DFA_STRING

(12) DFA_ASSIGN

(13) DFA_EQUAL

(14) DFA_NOT

(15) DFA_GT

(16) DFA_LT

(17) DFA_DONE

对应的 DFA 如图1所示：

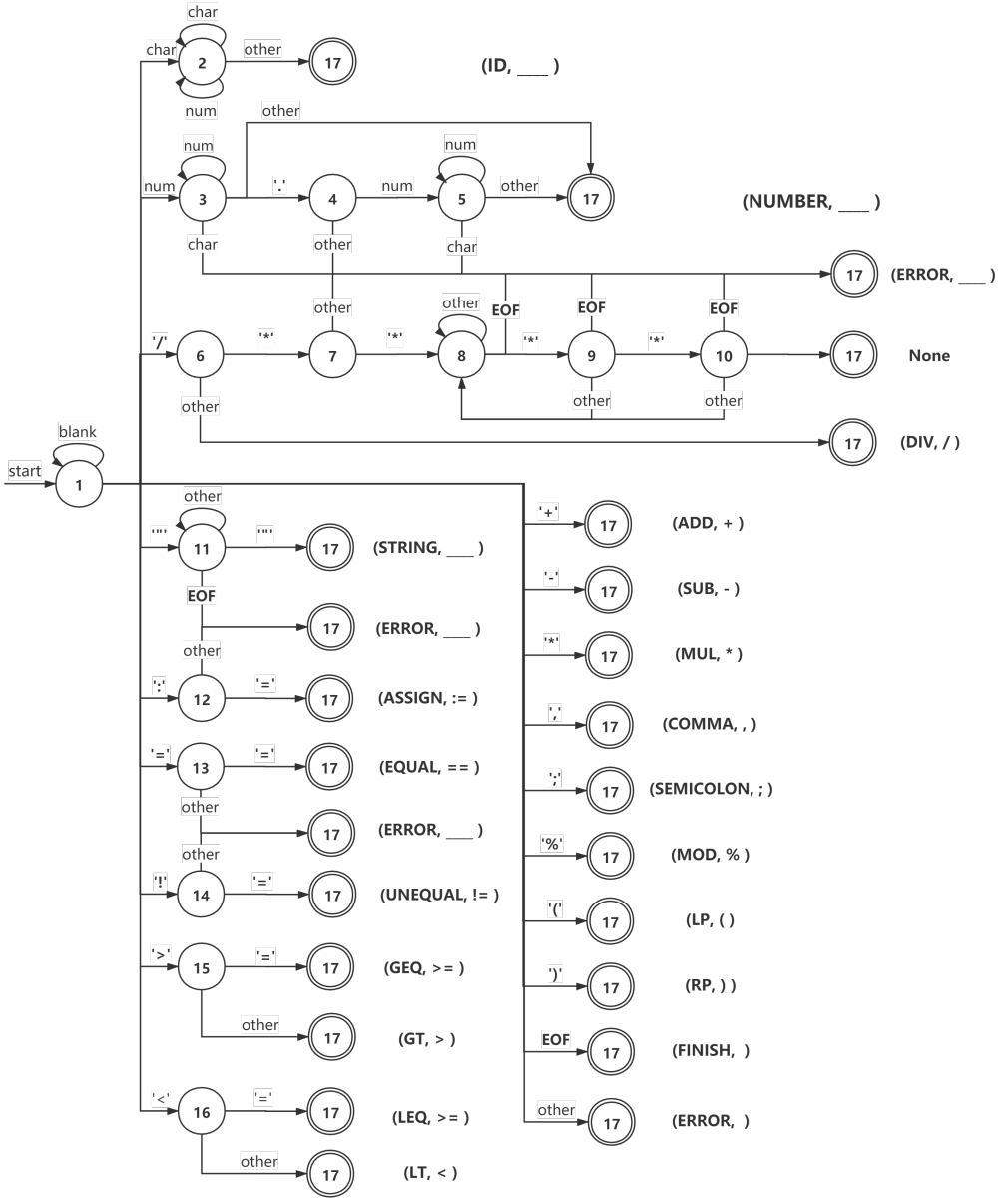


图 1: DFA 状态图

图中 blank 表示输入空白符（如空格、换行等），char 表示输入为字母，num 表示输入为阿拉伯数字，加粗的带引号的符号表示输入为相应符号，**EOF** 表示输入为文件结束符，other 表示输入为该状态下已有确定状态转换的输入字符以外的字符。当输出为 **None** 时表示为注释，没有任何 TOKEN 生成。当输出为 (**ERROR**, _____) 时，说明词法分析出现错误。当输出为 (**FINISH**, _____) 时，说明整个程序已经分析完毕。需要注意的是，为了实现方便，在 DFA 中我把标识符和关键字整合在了一起。输出 (**ID**, _____) 后，进一步地判断其是描述符还是关键字，再进行相应转换。

当返回的 TOKEN 类型为 ERROR 时，终止分析并报出对应错误。当返回的 TOKEN 类型为 FINISH 时，说明源程序翻译完毕，终止分析。关键实现代码与相关说明见5.1，具体实现代码见 code/myLexer.h 和 code/myLexer.c。

4.4 语法分析器

语法分析器负责将词法分析器输出的 TOKEN 序列转换成语法分析树，并检查语法错误。

本次实验中，我使用自顶向下的递归下降法分析技术实现语法分析。每个非终结符号有一个对应的过程。程序的执行从开始符号对应的过程开始，通过预测分析法向前看一个词法单元，确定当前非终结符号将要选择的产生式。通过不断递归非终结符号的过程，我们就可以得到一棵语法分析树。若在某个非终结符号中，发现无法 TOKEN 序列与任何产生式匹配，则进行回溯。若整个 TOKEN 序列被完整扫描并返回树根指针，则说明语法分析成功，否则我们需要定位具体错误位置并报错。

关键实现代码与相关说明见5.2，具体实现代码见 code/myParser.h 和 code/myParser.c。

5 关键实现代码与说明

5.1 词法分析 myLexer

在 myLexer.h 中，我定义了以下 TOKEN 类型：

```
typedef enum TokenType
{
    // 关键字
    IF,
    ELSE,
    WRITE,
    READ,
    RETURN,
    BEGIN,
    END,
    MAIN,
    INT,
    REAL,
    AND,
    OR,
    WHILE,
    BOOL,
    TRUE,
    FALSE,
```

```
    SEMICOLON,    // ;
    COMMA,        // ,
    LP,           // (
    RP,           // )
    ADD,          // +
    SUB,          // -
    MUL,          // *
    DIV,          // /
    MOD,          // %
    ASSIGN,       // :=
    LEQ,          // <=
    GEQ,          // >=
    LT,           // <
    GT,           // >
    EQUAL,        // ==
    UNEQUAL,      // !=
    NOT,          // !
    NUMBER,       // 数字
    ID,           // 描述符
    STRING,       // 字符串
    // 特殊type
    ERROR,
    FINISH
} TokenType;
```

类似的，我同样按照4.3所述定义了 DFA 状态：

```
typedef enum DFAStatus
{
    DFA_START,
    ......
    DFA_DONE
} DFAStatus;
```

TOKEN 结构如下所示：

```
typedef struct token
{
    enum TokenType type; // token类型
    char* str;           // token内容
    int RowIndex;        // 起始行号
    int ColIndex;        // 起始列号
} token;
```

相比普通的 TOKEN，我添加了 RowIndex 和 ColIndex 用于记录 TOKEN 在源程序的位置，便于后续定位错误位置。

用于从 TINY+ 源程序中获取 TOKEN 的函数如下所示：

```
// 获取下一个TOKEN
```

```
struct token* getNextToken(FILE* source, int* RowIndexPtr, int* ColIndexPtr) {
    // 创建并初始化TOKEN
    struct token* tp = (struct token*)calloc(1, sizeof(struct token));
    tp->str = (char*)calloc(MAXBUFLEN, sizeof(char));
    int StrIndex = 0;
    int StrLen = MAXBUFLEN;

    char ch;// 当前字符
    // 初始化DFA
    enum DFAStatus CurrentStatus = DFA_START;
    Bool isSave;// 是否存储该字符
    Bool isRecordLocation = False;// 是否在TOKEN中记录位置
    Bool isUpdateLocation = True;// 是否更新RowIndexPtr和ColIndexPtr存储的位置

    while (True) {
        // 获取下一个字符
        ch = fgetc(source);
        // 默认存储该字符
        isSave = True;
        // DFA状态跳转
        switch (CurrentStatus) {
        case DFA_START:
            if (isBlankChar(ch) != True)
                isRecordLocation = True;
            if (isBlankChar(ch) == True)
                isSave = False;
            else if (isDigit(ch) == True)
                CurrentStatus = DFA_INTNUM;
            else if (isAlpha(ch) == True)
                CurrentStatus = DFA_ID;
            else if (ch == '"')
                CurrentStatus = DFA_STRING;
            else if (ch == '/')
                CurrentStatus = DFA_DIV_OR_COMMENT;
            else if (ch == ':')
                CurrentStatus = DFA_ASSIGN;
            else if (ch == '=')
                CurrentStatus = DFA_EQUAL;
            else if (ch == '!')
                CurrentStatus = DFA_NOT;
            else if (ch == '>')
                CurrentStatus = DFA_GT;
            else if (ch == '<')
                CurrentStatus = DFA_LT;
            else {
                CurrentStatus = DFA_DONE;
                if (ch == ',')
                    tp->type = COMMA;
```

```
            else if (ch == ';')
                tp->type = SEMICOLON;
            ......
            else if (ch == EOF) {
                tp->type = FINISH;
                isSave = False;
            }
            else
                tp->type = ERROR;
        }
        break;
    ......
    default:
        CurrentStatus = DFA_DONE;
        tp->type = ERROR;
        break;
    }
    // 存储TOKEN起始位置
    if (isRecordLocation) {
        tp->ColIndex = *ColIndexPtr;
        tp->RowIndex = *RowIndexPtr;
        isRecordLocation = False;
    }
    // 更新计数器记录的位置
    if (isUpdateLocation) {
        if (ch == '\n')
        {
            (*RowIndexPtr) += 1;
            (*ColIndexPtr) = 1;
        }
        else
            (*ColIndexPtr) += 1;
    }
    // 存储字符
    if (isSave) {
        // 空间不足，开辟新空间
        if (StrIndex >= StrLen)
        {
            StrLen += MAXBUFLEN;
            tp->str = (char*)realloc(tp->str, StrLen * sizeof(char));
        }
        tp->str[StrIndex++] = ch;
    }
    // DFA到达结束状态，结束循环
    if (CurrentStatus == DFA_DONE) {
        // 存'\0'
        if (StrIndex >= StrLen) {
            StrLen += 1;
```

```
                tp->str = (char*)realloc(tp->str, StrLen * sizeof(char));
            }
            tp->str[StrIndex++] = '\0';
            // 若为错误，报错
            if (tp->type == ERROR)
                printf("\n[ERROR] %d:%d: %s\n", tp->RowIndex, tp->ColIndex, tp->str);
            // 将特定ID转换为关键字
            else if (tp->type == ID) {
                for (int i = 0; i < KeyTokensLen; i++) {
                    if (strcmp(tp->str, KeyTokens[i].str) == 0) {
                        tp->type = KeyTokens[i].type;
                        break;
                    }
                }
            }
            break;
        }
    }
    return tp;
}
```

关键部分为 while 语句中的 switch 语句，负责进行 DFA 状态跳转。DFA 到达结束状态后，再进行错误判断和关键字类型判断，最终输出结果

5.2 语法分析 myParser

语法分析树需要树节点来组成多叉树结构。在 myParser.h 中，我定义了如下节点结构：

```
// 节点
typedef struct TreeNode
{
    NodeType nodetype;// 节点类型
    struct TreeNode** ChildPtrList;// 子节点指针数组的指针
    int ChildCurrentIndex;// 当前子结点数量
    int ChildMaxNum;// 子结点指针数组支持的最大数量
    token* tp;// 叶子节点对应的TOKEN
} TreeNode;
```

节点类型如下所示。可见，我定义了各个非终结符号和终结符号作为节点类型，其中终结符号统一为 FACTOR。

```
// 节点类型
typedef enum NodeType
{
    NT_PROGRAM,
    NT_FUNCTION_DECL,
    NT_TYPE,
    NT_FORMAL_PARAMS,
    NT_FORMAL_PARAM,
```

```
    NT_BLOCK,
    NT_STATEMENT,
    NT_LOCAL_VAR_DECL,
    NT_ASSIGN_STMT,
    NT_RETURN_STMT,
    NT_IF_STMT,
    NT_WHILE_STMT,
    NT_WRITE_STMT,
    NT_READ_STMT,
    NT_EXPRESSION,
    NT_MULTIPLICATIVE_EXPR,
    NT_PRIMARY_EXPR,
    NT_ASSIGN_EXPR,
    NT_BOOL_MULTI_EXPR,
    NT_BOOL_EXPRESSION,
    NT_ACTUAL_PARAMS,
    NT_FACTOR,
} NodeType;
```

各个非终结符号对应的过程函数列表如下所示：

```
TreeNode* match(token** tpl, int* IndexPtr, int MaxTokenNum, TokenType type, Bool isThrow)
    ;
TreeNode* analyseProgram(token** tpl, int MaxTokenNum);
TreeNode* analyseFunctionDecl(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseType(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseFormalParams(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseFormalParam(token** tpl, int* IndexPtr, int MaxTokenNum);
Bool isFormalParam(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseBlock(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseStatement(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseLocalVarDecl(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseAssignStmt(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseReturnStmt(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseIfStmt(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseWhileStmt(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseWriteStmt(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseReadStmt(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseExpression(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseMultiplicativeExpr(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analysePrimaryExpr(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseAssignExpr(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseBoolMultiExpr(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseBoolExpression(token** tpl, int* IndexPtr, int MaxTokenNum);
TreeNode* analyseActualParams(token** tpl, int* IndexPtr, int MaxTokenNum);
```

函数接收 TOKEN 序列数组指针，通过 tpl[*IndexPtr] 获取下一个要分析的 TOKEN 的指针。MaxTokenNum 是 TOKEN 序列长度。match() 函数不是非终结符号对应的过程，其负责判断当前要分析的 TOKEN 是不是想要的 type，是的话就生成一个对应的叶子节点并返回。

由于相关函数数量过多，我们仅以 analyseFunctionDecl() 为例介绍，具体代码如下所示：

```
TreeNode* analyseFunctionDecl(token** tpl, int* IndexPtr, int MaxTokenNum) {
    // 创建新节点
    TreeNode* fp = buildTreeNode(False);
    int ChildIndex = 0;
    fp->nodetype = NT_FUNCTION_DECL;
    TreeNode* cp = NULL;
    // 尝试匹配非终结符号Type，若失败则报错返回空指针
    cp = analyseType(tpl, IndexPtr, MaxTokenNum);
    if (cp != NULL)
        addChildNode(fp, cp);
    else {
        if (*IndexPtr < MaxTokenNum)
            throwParserError(tpl[*IndexPtr], "Type doesn't finish.");
        else
            throwParserError(tpl[*IndexPtr - 1], "Type doesn't finish.");
        return NULL;
    }
    // 尝试匹配终结符号MAIN，允许失败
    cp = match(tpl, IndexPtr, MaxTokenNum, MAIN, False);
    if (cp != NULL)
        addChildNode(fp, cp);
    // 尝试匹配终结符号id，若失败则报错返回空指针
    cp = match(tpl, IndexPtr, MaxTokenNum, ID, True);
    if (cp != NULL)
        addChildNode(fp, cp);
    else
        return NULL;
    // 尝试匹配终结符号(，若失败则报错返回空指针
    cp = match(tpl, IndexPtr, MaxTokenNum, LP, True);
    if (cp != NULL)
        addChildNode(fp, cp);
    else
        return NULL;
    // 尝试匹配非终结符号FormalParams，若失败则报错返回空指针
    cp = analyseFormalParams(tpl, IndexPtr, MaxTokenNum);
    if (cp != NULL)
        addChildNode(fp, cp);
    else {
        if (*IndexPtr < MaxTokenNum)
            throwParserError(tpl[*IndexPtr], "FormalParams doesn't finish.");
        else
            throwParserError(tpl[*IndexPtr - 1], "FormalParams doesn't finish.");
        return NULL;
    }
    // 尝试匹配终结符号)，若失败则报错返回空指针
    cp = match(tpl, IndexPtr, MaxTokenNum, RP, True);
    if (cp != NULL)
```

```
48          addChildNode(fp, cp);
49      else
50          return NULL;
51      // 尝试匹配非终结符号Block，若失败则报错返回空指针
52      cp = analyseBlock(tpl, IndexPtr, MaxTokenNum);
53      if (cp != NULL)
54          addChildNode(fp, cp);
55      else {
56          if (*IndexPtr < MaxTokenNum)
57              throwParserError(tpl[*IndexPtr], "Block doesn't finish.");
58          else
59              throwParserError(tpl[*IndexPtr - 1], "Block doesn't finish.");
60          return NULL;
61      }
62      return fp;
63  }
```

其中与终结符号匹配的函数 match() 函数如下所示：

```
1   TreeNode* match(token** tpl, int* IndexPtr, int MaxTokenNum, TokenType type, Bool isThrow)
        {
2       TreeNode* np = NULL;
3       // 若匹配对应类型
4       if (*IndexPtr < MaxTokenNum && tpl[*IndexPtr]->type == type) {
5           // 创建新节点并赋值
6           np = buildTreeNode(True);
7           np->nodetype = NT_FACTOR;
8           np->tp = tpl[*IndexPtr];
9           // 偏移+1
10          (*IndexPtr)++;
11      }
12      // 不匹配，报错
13      if (isThrow && np == NULL) {
14          char output[MAXMESLEN];
15          if (*IndexPtr >= MaxTokenNum) {
16              snprintf(output, MAXMESLEN, "(%s,%s) Expect a %s.", getTokenName(tpl[*IndexPtr
                    - 1]->type), tpl[*IndexPtr - 1]->str, getTokenName(type));
17              throwParserError(tpl[*IndexPtr - 1], output);
18          }
19          else {
20              snprintf(output, MAXMESLEN, "(%s,%s) Expect a %s.", getTokenName(tpl[*IndexPtr
                    ]->type), tpl[*IndexPtr]->str, getTokenName(type));
21              throwParserError(tpl[*IndexPtr], output);
22          }
23      }
24      return np;
25  }
```

用于添加子节点的函数 addChildNode() 如下所示：

```
// 添加子节点
void addChildNode(TreeNode* fp, TreeNode* cp) {
    // 空间不够的话扩容
    if (fp->ChildCurrentIndex >= fp->ChildMaxNum) {
        fp->ChildMaxNum += CHILDMAXNUM;
        fp->ChildPtrList = (TreeNode**)realloc(fp->ChildPtrList, fp->ChildMaxNum * sizeof(
            TreeNode*));
        for (int i = fp->ChildCurrentIndex; i < fp->ChildMaxNum; i++)
            fp->ChildPtrList[i] = NULL;
    }
    // 添加
    fp->ChildPtrList[fp->ChildCurrentIndex++] = cp;
}
```

其余代码详见 code/myParser.h 和 code/myParser.c，在此不再赘述。

6 实验结果

6.1 词法分析

首先我们使用对没有 bug 的 TINY 源程序 good_example.tiny 进行词法分析。good_example.tiny 代码如下所示：

```
/** this is a comment line in the sample program **/
INT f2(INT x, INT y )
BEGIN
   INT z;
   z := x*x - y*y;
   RETURN z;
END
INT MAIN f1()
BEGIN
   INT x;
   READ(x, "A41.input");
   INT y;
   READ(y, "A42.input");
   INT z;
   z := f2(x,y) + f2(y,x);
   WRITE (z, "A4.output");
END
```

输出的 TOKEN 序列如下所示。由于序列过长，我们只展示部分内容，完整输出见8.1。

```
(INT, INT)
(ID, f2)
(LP, ()
(INT, INT)
(ID, x)
```

```
(COMMA, ,)
(INT, INT)
(ID, y)
(RP, ))
(BEGIN, BEGIN)
(INT, INT)
(ID, z)
(SEMICOLON, ;)
......
(WRITE, WRITE)
(LP, ()
(ID, z)
(COMMA, ,)
(STRING, "A4.output")
(RP, ))
(SEMICOLON, ;)
(END, END)
```

可见 TOKEN 序列正确，没有错误。

然后我们尝试对没有 bug 的 TINY+ 源程序 good_example.tinyplus 进行词法分析。good_example.tinyplus 代码如下所示：

```
/** This is a TINY+ program without bugs. **/
INT f2(INT x, INT y )
BEGIN
    INT a;
    a := 10;
    WHILE (a >= x)
        a := a % 2;
    RETURN a+y;
END
INT MAIN f1()
BEGIN
    BOOL flag:=TRUE;
    INT x:=5, y;
    REAL c:=4.521;
    IF (flag)
        x := (x+3)*4;
    ELSE
    BEGIN
        READ(y, "input y");
        c := y - 3;
    END
    INT z:= f2(x,y) + f2(y,x);
    WRITE (z, "output z");
END
```

输出的 TOKEN 序列如下所示。由于序列过长，我们只展示部分内容，完整输出见8.2。

```
(INT, INT)
(ID, f2)
(LP, ()
(INT, INT)
(ID, x)
(COMMA, ,)
(INT, INT)
(ID, y)
(RP, ))
(BEGIN, BEGIN)
(INT, INT)
(ID, a)
(SEMICOLON, ;)
(ID, a)
(ASSIGN, :=)
(NUMBER, 10)
(SEMICOLON, ;)
......
(WRITE, WRITE)
(LP, ()
(ID, z)
(COMMA, ,)
(STRING, "output z")
(RP, ))
(SEMICOLON, ;)
(END, END)
```

可见 TOKEN 序列正确，没有错误。

词法分析器具有检查词法错误的能力。bug_example.tinyplus 代码如下所示：

```
/** This is a TINY+ program with bug(s). **/
INT MAIN f()
BEGIN
    BOOL flag:=TRUE;
    INT 3x:=5;
    IF (flag)
        x := (x+3)*4;
    WRITE (x, "output x");
END
```

可见第 5 行代码中，出现了以数字开头的错误标识符 3x。

词法分析器输出如下所示：

```
[INFO] Lexical analysis:
(INT, INT)
(MAIN, MAIN)
(ID, f)
(LP, ()
(RP, ))
```

```
(BEGIN, BEGIN)
(BOOL, BOOL)
(ID, flag)
(ASSIGN, :=)
(TRUE, TRUE)
(SEMICOLON, ;)
(INT, INT)

[ERROR] 5:9: 3x

[INFO] Lexical analysis interrupted.
```

可见程序找到了错误。

另一个有 bug 的 bug_example2.tinyplus 代码如下所示：

```
/* This is a TINY+ program with bug(s). **/
INT MAIN f()
BEGIN
    BOOL flag:=TRUE;
    INT 3x:=5;
    IF (flag)
        x := (x+3)*4;
    WRITE (x, "output x");
END
```

可见第 1 行代码中，“/**” 漏了一个 “*”。

词法分析器输出如下所示：

```
[INFO] Lexical analysis:

[ERROR] 1:1: /*

[INFO] Lexical analysis interrupted.
```

可见程序找到了错误。

6.2 语法分析

同样，我们首先对 good_example.tiny 进行语法分析。语法分析树显示如图2所示。由于有些分支过于复杂，我们省略了部分显示，完整输出结果见8.1。

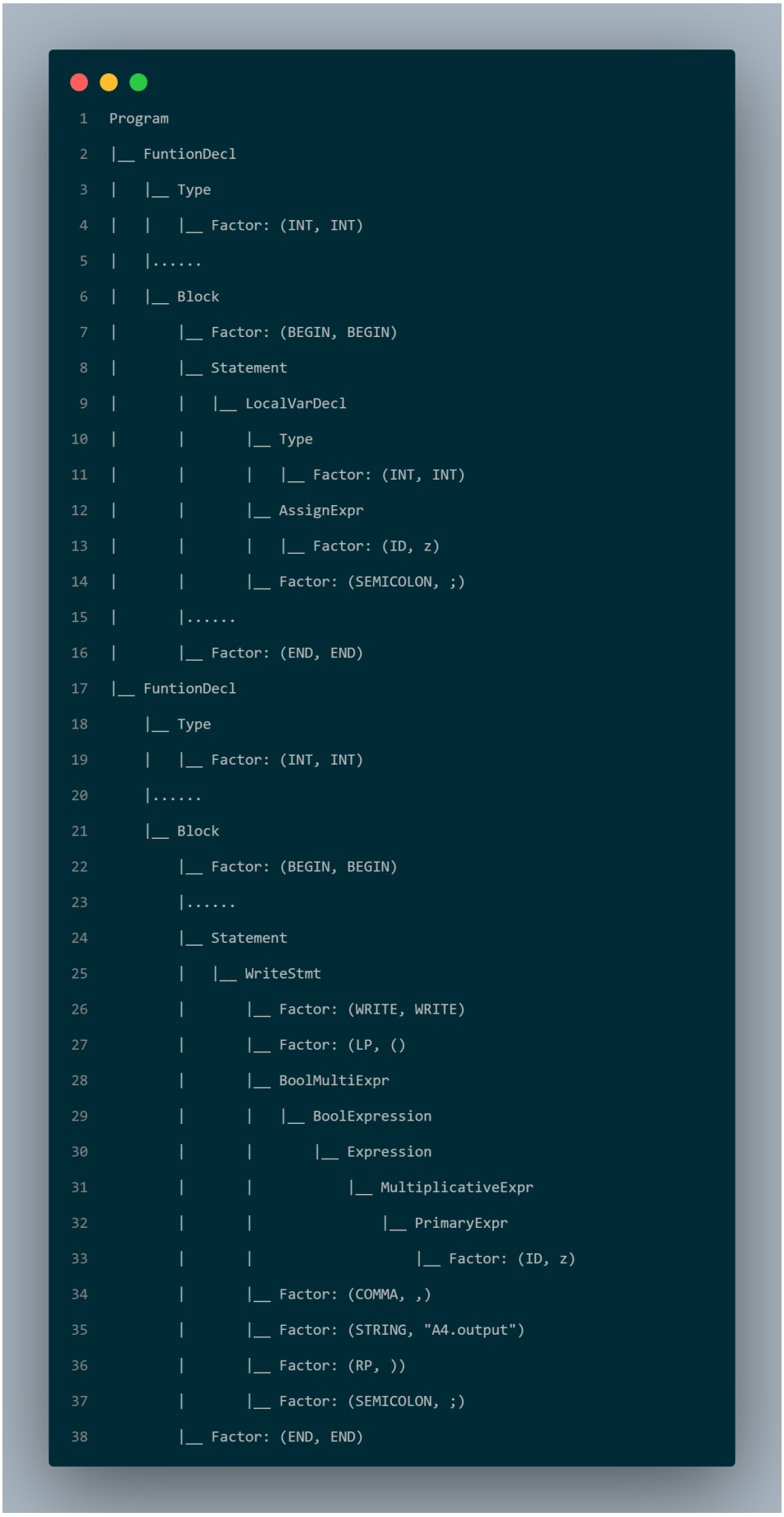
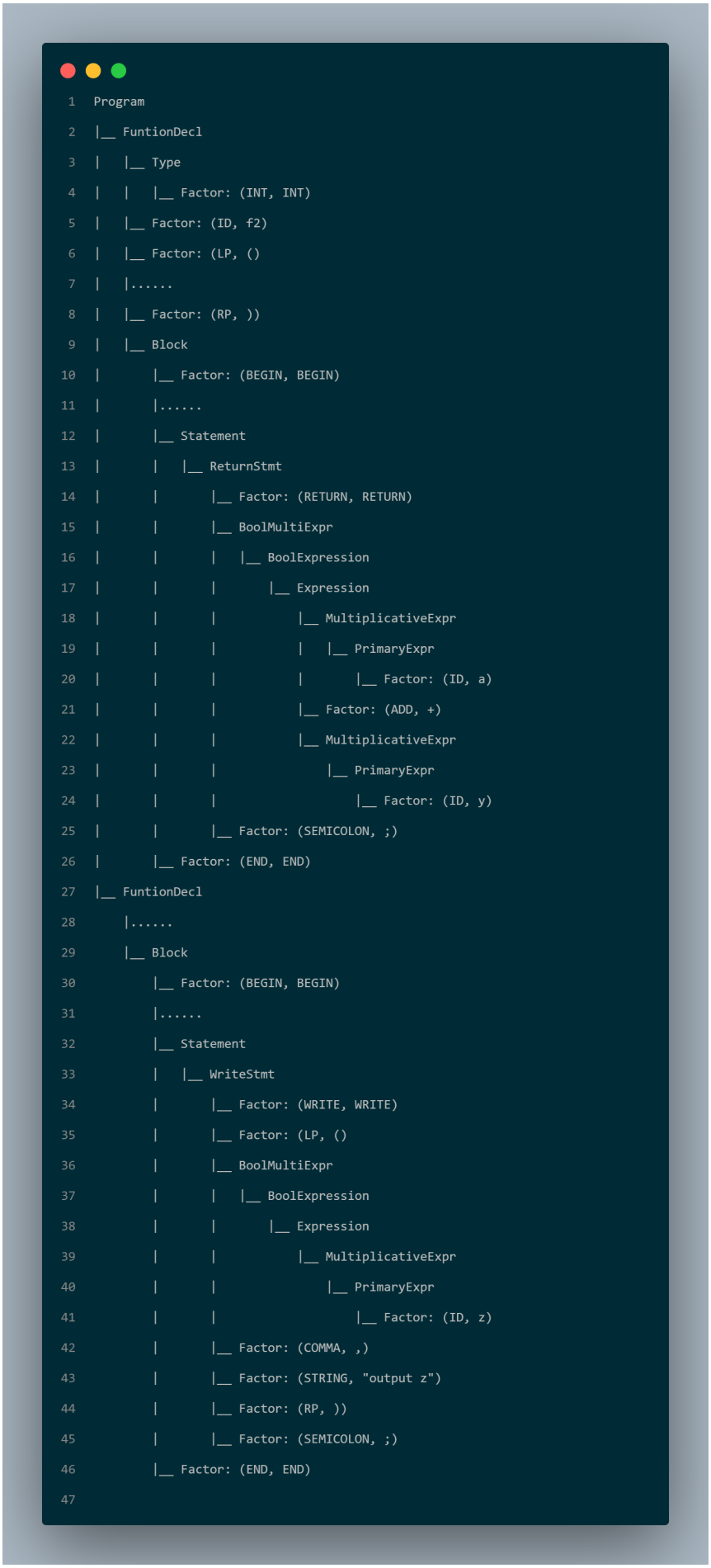


图 2: good_example.tiny 语法树

然后我们对 good_example.tinyplus 进行语法分析。语法分析树显示如图3所示。完整输出结果见8.2。



```
Program
|__ FuntionDecl
|   |__ Type
|   |   |__ Factor: (INT, INT)
|   |__ Factor: (ID, f2)
|   |__ Factor: (LP, ()
|   |......
|   |__ Factor: (RP, ))
|   |__ Block
|       |__ Factor: (BEGIN, BEGIN)
|       |......
|       |__ Statement
|       |   |__ ReturnStmt
|       |       |__ Factor: (RETURN, RETURN)
|       |       |__ BoolMultiExpr
|       |       |   |__ BoolExpression
|       |       |       |__ Expression
|       |       |           |__ MultiplicativeExpr
|       |       |           |   |__ PrimaryExpr
|       |       |           |       |__ Factor: (ID, a)
|       |       |           |__ Factor: (ADD, +)
|       |       |           |__ MultiplicativeExpr
|       |       |               |__ PrimaryExpr
|       |       |                   |__ Factor: (ID, y)
|       |       |__ Factor: (SEMICOLON, ;)
|       |__ Factor: (END, END)
|__ FuntionDecl
    |......
    |__ Block
        |__ Factor: (BEGIN, BEGIN)
        |......
        |__ Statement
        |   |__ WriteStmt
        |       |__ Factor: (WRITE, WRITE)
        |       |__ Factor: (LP, ()
        |       |__ BoolMultiExpr
        |       |   |__ BoolExpression
        |       |       |__ Expression
        |       |           |__ MultiplicativeExpr
        |       |               |__ PrimaryExpr
        |       |                   |__ Factor: (ID, z)
        |       |__ Factor: (COMMA, ,)
        |       |__ Factor: (STRING, "output z")
        |       |__ Factor: (RP, ))
        |       |__ Factor: (SEMICOLON, ;)
        |__ Factor: (END, END)
```

图 3: good_example2.tinyplus 语法树

语法分析器也可以检查一定的语法错误。错误代码 bug_example3.tinyplus 如下所示：

```
1 /** This is a TINY+ program with bug(s). **/
2 INT MAIN f()
3 BEGIN
4     BOOL flag:=TRUE;
5     INT x==5;
6     IF (flag)
7         x := (x+3)*4;
8     WRITE (x, "output x");
9 END
```

可以发现，第 5 行中我们在赋值语句中错误使用了 “==”。

语法分析输出结果如下所示：

```
1 [INFO] Parser analysis:
2
3 [ERROR] 5:10: (EQUAL,==) Expect a SEMICOLON.
4 [ERROR] 5:10: LocalVarDecl doesn't finish.
5 [ERROR] 5:10: Statement doesn't finish.
6 [ERROR] 5:10: Block doesn't finish.
7 [ERROR] 5:10: EunctionDecl doesn't finish.
8
9 [INFO] Parser analysis interrupted.
```

可见，语法分析器找到了这个错误，并给出了修改建议。

另一个错误代码 bug_example4.tinyplus 如下所示：

```
1 /** This is a TINY+ program with bug(s). **/
2 INT MAIN f()
3 BEGIN
4     BOOL flag:=TRUE;
5     INT x:=5;
6     IF (TRUE)
7         x := (x+3)*4;
8     WRITE (x, "output x")
9 END
```

可以发现，第 8 行中我们在语句的最后遗漏了分号。

语法分析输出结果如下所示：

```
1 [INFO] Parser analysis:
2
3 [ERROR] 9:1: (END,END) Expect a SEMICOLON.
4 [ERROR] 9:1: WriteStmt doesn't finish.
5 [ERROR] 9:1: Statement doesn't finish.
6 [ERROR] 9:1: Block doesn't finish.
7 [ERROR] 9:1: EunctionDecl doesn't finish.
8
```

```
[INFO] Parser analysis interrupted.
```

可见，语法分析器发现没有分号，并给出了修改建议。

7 实验总结

本次实验是编译原理课程到目前为止难度最高的一次实验。我前后花了一星期多的时间来完成这次实验，差不多写了两千行代码，最终完成对 TINY+ 语言的词法分析和语法分析。虽然这次实验看起来难度很大，但一旦了解词法分析和语法分析的方法后，实现的难度也就不那么高了。此外，在 TINY+ 中，我加入了很多新支持，有些很实用，也有一些就纯属有趣了。

本次实验中，最让我印象深刻的是词法分析中 DFA 的构造。一开始我没有画出 DFA 转换图，而是直接动手开干。但后来撰写实验报告时我认真地画了次 DFA 图，然后对照代码发现我的程序有很多漏洞。因为不清楚各个状态之间的转换，我的程序遗漏了很多输入特例，这就导致很容易在面对异常输入时出现 bug。后来我解决了这个问题。

由于时间有限，我没有给程序加入语义分析功能，而只是单纯的词法和语法检查，未免有些遗憾。希望我能顺利完成下一次实验。

8 附录

8.1 good_example.tiny 完整分析输出

```
-----------------------------------------
|   TINY+ v0.4 :: 18308013 ChenJiahao   |
-----------------------------------------
[INFO] Welcome to use TINY+ compiler!
[INFO] Please input your source file full path:
[INPUT] E:\code\Windows\C++\Compiler_Principle\ex3\my_compiler\good_sample.tiny
[INFO] Lexical analysis:

(INT, INT)
(ID, f2)
(LP, ()
(INT, INT)
(ID, x)
(COMMA, ,)
(INT, INT)
(ID, y)
(RP, ))
(BEGIN, BEGIN)
(INT, INT)
(ID, z)
(SEMICOLON, ;)
(ID, z)
(ASSIGN, :=)
```

```
(ID, x)
(MUL, *)
(ID, x)
(SUB, -)
(ID, y)
(MUL, *)
(ID, y)
(SEMICOLON, ;)
(RETURN, RETURN)
(ID, z)
(SEMICOLON, ;)
(END, END)
(INT, INT)
(MAIN, MAIN)
(ID, f1)
(LP, ()
(RP, ))
(BEGIN, BEGIN)
(INT, INT)
(ID, x)
(SEMICOLON, ;)
(READ, READ)
(LP, ()
(ID, x)
(COMMA, ,)
(STRING, "A41.input")
(RP, ))
(SEMICOLON, ;)
(INT, INT)
(ID, y)
(SEMICOLON, ;)
(READ, READ)
(LP, ()
(ID, y)
(COMMA, ,)
(STRING, "A42.input")
(RP, ))
(SEMICOLON, ;)
(INT, INT)
(ID, z)
(SEMICOLON, ;)
(ID, z)
(ASSIGN, :=)
(ID, f2)
(LP, ()
(ID, x)
(COMMA, ,)
(ID, y)
```

```
(RP, ))
(ADD, +)
(ID, f2)
(LP, ()
(ID, y)
(COMMA, ,)
(ID, x)
(RP, ))
(SEMICOLON, ;)
(WRITE, WRITE)
(LP, ()
(ID, z)
(COMMA, ,)
(STRING, "A4.output")
(RP, ))
(SEMICOLON, ;)
(END, END)

[INFO] Lexical analysis Finished.
[INFO] Parser analysis:

Program
|__ FuntionDecl
|    |__ Type
|    |    |__ Factor: (INT, INT)
|    |__ Factor: (ID, f2)
|    |__ Factor: (LP, ()
|    |__ FormalParams
|    |    |__ FormalParam
|    |    |    |__ Type
|    |    |    |    |__ Factor: (INT, INT)
|    |    |    |__ Factor: (ID, x)
|    |    |__ Factor: (COMMA, ,)
|    |    |__ FormalParam
|    |         |__ Type
|    |         |    |__ Factor: (INT, INT)
|    |         |__ Factor: (ID, y)
|    |__ Factor: (RP, ))
|    |__ Block
|         |__ Factor: (BEGIN, BEGIN)
|         |__ Statement
|         |    |__ LocalVarDecl
|         |         |__ Type
|         |         |    |__ Factor: (INT, INT)
|         |         |__ AssignExpr
|         |         |    |__ Factor: (ID, z)
|         |         |__ Factor: (SEMICOLON, ;)
|         |__ Statement
```

```
120 |       |   |__ AssignStmt
121 |       |       |__ AssignExpr
122 |       |       |   |__ Factor: (ID, z)
123 |       |       |   |__ Factor: (ASSIGN, :=)
124 |       |       |   |__ BoolMultiExpr
125 |       |       |       |__ BoolExpression
126 |       |       |           |__ Expression
127 |       |       |               |__ MultiplicativeExpr
128 |       |       |               |   |__ PrimaryExpr
129 |       |       |               |   |   |__ Factor: (ID, x)
130 |       |       |               |   |__ Factor: (MUL, *)
131 |       |       |               |   |__ PrimaryExpr
132 |       |       |               |       |__ Factor: (ID, x)
133 |       |       |               |__ Factor: (SUB, -)
134 |       |       |               |__ MultiplicativeExpr
135 |       |       |                   |__ PrimaryExpr
136 |       |       |                   |   |__ Factor: (ID, y)
137 |       |       |                   |__ Factor: (MUL, *)
138 |       |       |                   |__ PrimaryExpr
139 |       |       |                       |__ Factor: (ID, y)
140 |       |       |__ Factor: (SEMICOLON, ;)
141 |       |__ Statement
142 |       |   |__ ReturnStmt
143 |       |       |__ Factor: (RETURN, RETURN)
144 |       |       |__ BoolMultiExpr
145 |       |       |   |__ BoolExpression
146 |       |       |       |__ Expression
147 |       |       |           |__ MultiplicativeExpr
148 |       |       |               |__ PrimaryExpr
149 |       |       |                   |__ Factor: (ID, z)
150 |       |       |__ Factor: (SEMICOLON, ;)
151 |       |__ Factor: (END, END)
152 |__ FuntionDecl
153     |__ Type
154     |   |__ Factor: (INT, INT)
155     |__ Factor: (MAIN, MAIN)
156     |__ Factor: (ID, f1)
157     |__ Factor: (LP, ()
158     |__ FormalParams
159     |__ Factor: (RP, ))
160     |__ Block
161         |__ Factor: (BEGIN, BEGIN)
162         |__ Statement
163         |   |__ LocalVarDecl
164         |       |__ Type
165         |       |   |__ Factor: (INT, INT)
166         |       |__ AssignExpr
167         |       |   |__ Factor: (ID, x)
```

```
168         |       |__ Factor: (SEMICOLON, ;)
169         |__ Statement
170         |   |__ ReadStmt
171         |       |__ Factor: (READ, READ)
172         |       |__ Factor: (LP, ()
173         |       |__ Factor: (ID, x)
174         |       |__ Factor: (COMMA, ,)
175         |       |__ Factor: (STRING, "A41.input")
176         |       |__ Factor: (RP, ))
177         |       |__ Factor: (SEMICOLON, ;)
178         |__ Statement
179         |   |__ LocalVarDecl
180         |       |__ Type
181         |       |   |__ Factor: (INT, INT)
182         |       |__ AssignExpr
183         |       |   |__ Factor: (ID, y)
184         |       |__ Factor: (SEMICOLON, ;)
185         |__ Statement
186         |   |__ ReadStmt
187         |       |__ Factor: (READ, READ)
188         |       |__ Factor: (LP, ()
189         |       |__ Factor: (ID, y)
190         |       |__ Factor: (COMMA, ,)
191         |       |__ Factor: (STRING, "A42.input")
192         |       |__ Factor: (RP, ))
193         |       |__ Factor: (SEMICOLON, ;)
194         |__ Statement
195         |   |__ LocalVarDecl
196         |       |__ Type
197         |       |   |__ Factor: (INT, INT)
198         |       |__ AssignExpr
199         |       |   |__ Factor: (ID, z)
200         |       |__ Factor: (SEMICOLON, ;)
201         |__ Statement
202         |   |__ AssignStmt
203         |       |__ AssignExpr
204         |       |   |__ Factor: (ID, z)
205         |       |   |__ Factor: (ASSIGN, :=)
206         |       |   |__ BoolMultiExpr
207         |       |       |__ BoolExpression
208         |       |           |__ Expression
209         |       |               |__ MultiplicativeExpr
210         |       |               |   |__ PrimaryExpr
211         |       |               |       |__ Factor: (ID, f2)
212         |       |               |       |__ Factor: (LP, ()
213         |       |               |       |__ ActualParams
214         |       |               |       |   |__ Expression
215         |       |               |       |   |   |__ MultiplicativeExpr
```

```
        |       |               |       |   |       |__ PrimaryExpr
        |       |               |       |   |           |__ Factor: (ID, x)
        |       |               |       |   |__ Factor: (COMMA, ,)
        |       |               |       |   |__ Expression
        |       |               |       |       |__ MultiplicativeExpr
        |       |               |       |           |__ PrimaryExpr
        |       |               |       |               |__ Factor: (ID, y)
        |       |               |       |__ Factor: (RP, ))
        |       |               |__ Factor: (ADD, +)
        |       |               |__ MultiplicativeExpr
        |       |                   |__ PrimaryExpr
        |       |                       |__ Factor: (ID, f2)
        |       |                       |__ Factor: (LP, ()
        |       |                       |__ ActualParams
        |       |                       |   |__ Expression
        |       |                       |   |   |__ MultiplicativeExpr
        |       |                       |   |       |__ PrimaryExpr
        |       |                       |   |           |__ Factor: (ID, y)
        |       |                       |   |__ Factor: (COMMA, ,)
        |       |                       |   |__ Expression
        |       |                       |       |__ MultiplicativeExpr
        |       |                       |           |__ PrimaryExpr
        |       |                       |               |__ Factor: (ID, x)
        |       |                       |__ Factor: (RP, ))
        |       |__ Factor: (SEMICOLON, ;)
        |__ Statement
        |   |__ WriteStmt
        |       |__ Factor: (WRITE, WRITE)
        |       |__ Factor: (LP, ()
        |       |__ BoolMultiExpr
        |       |   |__ BoolExpression
        |       |       |__ Expression
        |       |           |__ MultiplicativeExpr
        |       |               |__ PrimaryExpr
        |       |                   |__ Factor: (ID, z)
        |       |__ Factor: (COMMA, ,)
        |       |__ Factor: (STRING, "A4.output")
        |       |__ Factor: (RP, ))
        |       |__ Factor: (SEMICOLON, ;)
        |__ Factor: (END, END)

[INFO] Parser analysis Finished.
```

8.2 good_example.tinyplus 完整分析输出

```
-----------------------------------------
|   TINY+ v0.4 :: 18308013 ChenJiahao   |
```

```
-----------------------------------------------
[INFO] Welcome to use TINY+ compiler!
[INFO] Please input your source file full path:
[INPUT] E:\code\Windows\C++\Compiler_Principle\ex3\my_compiler\good_sample.tinyplus
[INFO] Lexical analysis:

(INT, INT)
(ID, f2)
(LP, ()
(INT, INT)
(ID, x)
(COMMA, ,)
(INT, INT)
(ID, y)
(RP, ))
(BEGIN, BEGIN)
(INT, INT)
(ID, a)
(SEMICOLON, ;)
(ID, a)
(SEMICOLON, ;)
(ID, a)
(ASSIGN, :=)
(NUMBER, 10)
(SEMICOLON, ;)
(WHILE, WHILE)
(LP, ()
(ID, a)
(GEQ, >=)
(ID, x)
(RP, ))
(ID, a)
(ASSIGN, :=)
(ID, a)
(MOD, %)
(NUMBER, 2)
(SEMICOLON, ;)
(RETURN, RETURN)
(ID, a)
(ADD, +)
(ID, y)
(SEMICOLON, ;)
(END, END)
(INT, INT)
(MAIN, MAIN)
(ID, f1)
(LP, ()
(RP, ))
```

```
(BEGIN, BEGIN)
(BOOL, BOOL)
(ID, flag)
(ASSIGN, :=)
(TRUE, TRUE)
(SEMICOLON, ;)
(INT, INT)
(ID, x)
(ASSIGN, :=)
(NUMBER, 5)
(COMMA, ,)
(ID, y)
(SEMICOLON, ;)
(REAL, REAL)
(ID, c)
(ASSIGN, :=)
(NUMBER, 4.521)
(SEMICOLON, ;)
(IF, IF)
(LP, ()
(ID, flag)
(RP, ))
(ID, x)
(ASSIGN, :=)
(LP, ()
(ID, x)
(ADD, +)
(NUMBER, 3)
(RP, ))
(MUL, *)
(NUMBER, 4)
(SEMICOLON, ;)
(ELSE, ELSE)
(BEGIN, BEGIN)
(READ, READ)
(LP, ()
(ID, y)
(COMMA, ,)
(STRING, "input y")
(RP, ))
(SEMICOLON, ;)
(ID, c)
(ASSIGN, :=)
(ID, y)
(SUB, -)
(NUMBER, 3)
(SEMICOLON, ;)
(END, END)
```

```
(INT, INT)
(ID, z)
(ASSIGN, :=)
(ID, f2)
(LP, ()
(ID, x)
(COMMA, ,)
(ID, y)
(RP, ))
(ADD, +)
(ID, f2)
(LP, ()
(ID, y)
(COMMA, ,)
(ID, x)
(RP, ))
(SEMICOLON, ;)
(WRITE, WRITE)
(LP, ()
(ID, z)
(COMMA, ,)
(STRING, "output z")
(RP, ))
(SEMICOLON, ;)
(END, END)

[INFO] Lexical analysis Finished.
[INFO] Parser analysis:

Program
|__ FuntionDecl
|    |__ Type
|    |    |__ Factor: (INT, INT)
|    |__ Factor: (ID, f2)
|    |__ Factor: (LP, ()
|    |__ FormalParams
|    |    |__ FormalParam
|    |    |    |__ Type
|    |    |    |    |__ Factor: (INT, INT)
|    |    |    |__ Factor: (ID, x)
|    |    |__ Factor: (COMMA, ,)
|    |    |__ FormalParam
|    |         |__ Type
|    |         |    |__ Factor: (INT, INT)
|    |         |__ Factor: (ID, y)
|    |__ Factor: (RP, ))
|    |__ Block
|         |__ Factor: (BEGIN, BEGIN)
```

```
|       |__ Statement
|       |   |__ LocalVarDecl
|       |       |__ Type
|       |       |   |__ Factor: (INT, INT)
|       |       |__ AssignExpr
|       |       |   |__ Factor: (ID, a)
|       |       |__ Factor: (SEMICOLON, ;)
|       |__ Statement
|       |   |__ AssignStmt
|       |       |__ AssignExpr
|       |       |   |__ Factor: (ID, a)
|       |       |__ Factor: (SEMICOLON, ;)
|       |__ Statement
|       |   |__ AssignStmt
|       |       |__ AssignExpr
|       |       |   |__ Factor: (ID, a)
|       |       |   |__ Factor: (ASSIGN, :=)
|       |       |   |__ BoolMultiExpr
|       |       |       |__ BoolExpression
|       |       |           |__ Expression
|       |       |               |__ MultiplicativeExpr
|       |       |                   |__ PrimaryExpr
|       |       |                       |__ Factor: (NUMBER, 10)
|       |       |__ Factor: (SEMICOLON, ;)
|       |__ Statement
|       |   |__ WhileStmt
|       |       |__ Factor: (WHILE, WHILE)
|       |       |__ Factor: (LP, ()
|       |       |__ BoolMultiExpr
|       |       |   |__ BoolExpression
|       |       |       |__ Expression
|       |       |       |   |__ MultiplicativeExpr
|       |       |       |       |__ PrimaryExpr
|       |       |       |           |__ Factor: (ID, a)
|       |       |       |__ Factor: (GEQ, >=)
|       |       |       |__ Expression
|       |       |           |__ MultiplicativeExpr
|       |       |               |__ PrimaryExpr
|       |       |                   |__ Factor: (ID, x)
|       |       |__ Factor: (RP, ))
|       |       |__ Statement
|       |           |__ AssignStmt
|       |               |__ AssignExpr
|       |               |   |__ Factor: (ID, a)
|       |               |   |__ Factor: (ASSIGN, :=)
|       |               |   |__ BoolMultiExpr
|       |               |       |__ BoolExpression
|       |               |           |__ Expression
```

```
|       |               |               |__ MultiplicativeExpr
|       |               |                   |__ PrimaryExpr
|       |               |                   |   |__ Factor: (ID, a)
|       |               |                   |__ Factor: (MOD, %)
|       |               |                   |__ PrimaryExpr
|       |               |                       |__ Factor: (NUMBER, 2)
|       |               |__ Factor: (SEMICOLON, ;)
|       |__ Statement
|       |   |__ ReturnStmt
|       |       |__ Factor: (RETURN, RETURN)
|       |       |__ BoolMultiExpr
|       |       |   |__ BoolExpression
|       |       |       |__ Expression
|       |       |           |__ MultiplicativeExpr
|       |       |           |   |__ PrimaryExpr
|       |       |           |       |__ Factor: (ID, a)
|       |       |           |__ Factor: (ADD, +)
|       |       |           |__ MultiplicativeExpr
|       |       |               |__ PrimaryExpr
|       |       |                   |__ Factor: (ID, y)
|       |       |__ Factor: (SEMICOLON, ;)
|       |__ Factor: (END, END)
|__ FuntionDecl
    |__ Type
    |   |__ Factor: (INT, INT)
    |__ Factor: (MAIN, MAIN)
    |__ Factor: (ID, f1)
    |__ Factor: (LP, ()
    |__ FormalParams
    |__ Factor: (RP, ))
    |__ Block
        |__ Factor: (BEGIN, BEGIN)
        |__ Statement
        |   |__ LocalVarDecl
        |       |__ Type
        |       |   |__ Factor: (BOOL, BOOL)
        |       |__ AssignExpr
        |       |   |__ Factor: (ID, flag)
        |       |   |__ Factor: (ASSIGN, :=)
        |       |   |__ BoolMultiExpr
        |       |       |__ BoolExpression
        |       |           |__ Expression
        |       |               |__ MultiplicativeExpr
        |       |                   |__ PrimaryExpr
        |       |                       |__ Factor: (TRUE, TRUE)
        |       |__ Factor: (SEMICOLON, ;)
        |__ Statement
        |   |__ LocalVarDecl
```

```
243         |         |__ Type
244         |         |    |__ Factor: (INT, INT)
245         |         |__ AssignExpr
246         |         |    |__ Factor: (ID, x)
247         |         |    |__ Factor: (ASSIGN, :=)
248         |         |    |__ BoolMultiExpr
249         |         |        |__ BoolExpression
250         |         |            |__ Expression
251         |         |                |__ MultiplicativeExpr
252         |         |                    |__ PrimaryExpr
253         |         |                        |__ Factor: (NUMBER, 5)
254         |         |__ Factor: (COMMA, ,)
255         |         |__ AssignExpr
256         |         |    |__ Factor: (ID, y)
257         |         |__ Factor: (SEMICOLON, ;)
258         |__ Statement
259         |    |__ LocalVarDecl
260         |         |__ Type
261         |         |    |__ Factor: (REAL, REAL)
262         |         |__ AssignExpr
263         |         |    |__ Factor: (ID, c)
264         |         |    |__ Factor: (ASSIGN, :=)
265         |         |    |__ BoolMultiExpr
266         |         |        |__ BoolExpression
267         |         |            |__ Expression
268         |         |                |__ MultiplicativeExpr
269         |         |                    |__ PrimaryExpr
270         |         |                        |__ Factor: (NUMBER, 4.521)
271         |         |__ Factor: (SEMICOLON, ;)
272         |__ Statement
273         |    |__ IfStmt
274         |         |__ Factor: (IF, IF)
275         |         |__ Factor: (LP, ()
276         |         |__ BoolMultiExpr
277         |         |    |__ BoolExpression
278         |         |        |__ Expression
279         |         |            |__ MultiplicativeExpr
280         |         |                |__ PrimaryExpr
281         |         |                    |__ Factor: (ID, flag)
282         |         |__ Factor: (RP, ))
283         |         |__ Statement
284         |         |    |__ AssignStmt
285         |         |        |__ AssignExpr
286         |         |        |    |__ Factor: (ID, x)
287         |         |        |    |__ Factor: (ASSIGN, :=)
288         |         |        |    |__ BoolMultiExpr
289         |         |        |        |__ BoolExpression
290         |         |        |            |__ Expression
```

```
|       |       |                   |__ MultiplicativeExpr
|       |       |                       |__ PrimaryExpr
|       |       |                       |   |__ Factor: (LP, ()
|       |       |                       |   |__ BoolMultiExpr
|       |       |                       |   |   |__ BoolExpression
|       |       |                       |   |       |__ Expression
|       |       |                       |   |           |__ MultiplicativeExpr
|       |       |                       |   |           |   |__ PrimaryExpr
|       |       |                       |   |           |       |__ Factor: (ID, x)
|       |       |                       |   |           |__ Factor: (ADD, +)
|       |       |                       |   |           |__ MultiplicativeExpr
|       |       |                       |   |               |__ PrimaryExpr
|       |       |                       |   |                   |__ Factor: (NUMBER,
        3)
|       |       |                       |   |__ Factor: (RP, ))
|       |       |                       |__ Factor: (MUL, *)
|       |       |                       |__ PrimaryExpr
|       |       |                           |__ Factor: (NUMBER, 4)
|       |       |__ Factor: (SEMICOLON, ;)
|       |__ Factor: (ELSE, ELSE)
|       |__ Statement
|           |__ Block
|               |__ Factor: (BEGIN, BEGIN)
|               |__ Statement
|               |   |__ ReadStmt
|               |       |__ Factor: (READ, READ)
|               |       |__ Factor: (LP, ()
|               |       |__ Factor: (ID, y)
|               |       |__ Factor: (COMMA, ,)
|               |       |__ Factor: (STRING, "input y")
|               |       |__ Factor: (RP, ))
|               |       |__ Factor: (SEMICOLON, ;)
|               |__ Statement
|               |   |__ AssignStmt
|               |       |__ AssignExpr
|               |       |   |__ Factor: (ID, c)
|               |       |   |__ Factor: (ASSIGN, :=)
|               |       |   |__ BoolMultiExpr
|               |       |       |__ BoolExpression
|               |       |           |__ Expression
|               |       |               |__ MultiplicativeExpr
|               |       |               |   |__ PrimaryExpr
|               |       |               |       |__ Factor: (ID, y)
|               |       |               |__ Factor: (SUB, -)
|               |       |               |__ MultiplicativeExpr
|               |       |                   |__ PrimaryExpr
|               |       |                       |__ Factor: (NUMBER, 3)
|               |       |__ Factor: (SEMICOLON, ;)
```

```
|                   |__ Factor: (END, END)
|__ Statement
|   |__ LocalVarDecl
|       |__ Type
|       |   |__ Factor: (INT, INT)
|       |__ AssignExpr
|       |   |__ Factor: (ID, z)
|       |   |__ Factor: (ASSIGN, :=)
|       |   |__ BoolMultiExpr
|       |       |__ BoolExpression
|       |           |__ Expression
|       |               |__ MultiplicativeExpr
|       |               |   |__ PrimaryExpr
|       |               |       |__ Factor: (ID, f2)
|       |               |       |__ Factor: (LP, ()
|       |               |       |__ ActualParams
|       |               |       |   |__ Expression
|       |               |       |   |   |__ MultiplicativeExpr
|       |               |       |   |       |__ PrimaryExpr
|       |               |       |   |           |__ Factor: (ID, x)
|       |               |       |   |__ Factor: (COMMA, ,)
|       |               |       |   |__ Expression
|       |               |       |       |__ MultiplicativeExpr
|       |               |       |           |__ PrimaryExpr
|       |               |       |               |__ Factor: (ID, y)
|       |               |       |__ Factor: (RP, ))
|       |               |__ Factor: (ADD, +)
|       |               |__ MultiplicativeExpr
|       |                   |__ PrimaryExpr
|       |                       |__ Factor: (ID, f2)
|       |                       |__ Factor: (LP, ()
|       |                       |__ ActualParams
|       |                       |   |__ Expression
|       |                       |   |   |__ MultiplicativeExpr
|       |                       |   |       |__ PrimaryExpr
|       |                       |   |           |__ Factor: (ID, y)
|       |                       |   |__ Factor: (COMMA, ,)
|       |                       |   |__ Expression
|       |                       |       |__ MultiplicativeExpr
|       |                       |           |__ PrimaryExpr
|       |                       |               |__ Factor: (ID, x)
|       |                       |__ Factor: (RP, ))
|       |__ Factor: (SEMICOLON, ;)
|__ Statement
|   |__ WriteStmt
|       |__ Factor: (WRITE, WRITE)
|       |__ Factor: (LP, ()
|       |__ BoolMultiExpr
```

```
386         |       |   |__ BoolExpression
387         |       |       |__ Expression
388         |       |           |__ MultiplicativeExpr
389         |       |               |__ PrimaryExpr
390         |       |                   |__ Factor: (ID, z)
391         |       |__ Factor: (COMMA, ,)
392         |       |__ Factor: (STRING, "output z")
393         |       |__ Factor: (RP, ))
394         |       |__ Factor: (SEMICOLON, ;)
395         |__ Factor: (END, END)
396
397 [INFO] Parser analysis Finished.
```