

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	2
2	实验内容	2
3	实验要求	2
4	算法原理与描述	2
4.1	TINY 语言定义	2
4.1.1	TINY Lexicon	2
4.1.2	TINY Grammar	3
4.2	TINY+ 语言定义	4
4.2.1	TINY+ Lexicon	4
4.2.2	TINY+ Grammar	4
4.3	词法分析器	6
4.4	语法分析器	8
5	关键实现代码与说明	8
5.1	词法分析 myLexer	8
5.2	语法分析 myParser	12
6	实验结果	16
6.1	词法分析	16
6.2	语法分析	19
7	实验总结	23
8	附录	23
8.1	good_example.tiny 完整分析输出	23
8.2	good_example.tinyplus 完整分析输出	28

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; $(ex_1 \mid \dots \mid ex_n)$ 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 \ [\mathbf{MAIN}] \ \mathbf{id} \ '(\ [FormalParams] \)' \ Block \\ FormalParams &\rightarrow FormalParam \ (\ ',' \ FormalParam \)^* \\ FormalParam &\rightarrow Type \ \mathbf{id} \\ Type &\rightarrow \mathbf{INT} \mid \mathbf{REAL} \end{aligned}$$

对于 statements, 其 EBNF 描述如下:

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

对于 expressions, 其 EBNF 描述如下:

$$\begin{aligned} Expression &\rightarrow MultiplicativeExpr \ (\ ('+' \ | \ '-') \ MultiplicativeExpr \)^* \\ MultiplicativeExpr &\rightarrow PrimaryExpr \ (\ ('*' \ | \ '/') \ PrimaryExpr \)^* \\ PrimaryExpr &\rightarrow \mathbf{num} \\ &\quad | \ \mathbf{id} \\ &\quad | \ ' (\ Expression \) ' \\ &\quad | \ \mathbf{id} \ ' (\ [ActualParams] \) ' \\ BoolExpression &\rightarrow Expression \ (\ '!= ' \ | \ '== ') \ 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 \ [\ \mathbf{MAIN} \] \ \mathbf{id} \ ' (' \ [\ FormalParams \] \)' \ Block \\ FormalParams &\rightarrow FormalParam \ (\ ',' \ FormalParam \)^* \\ FormalParam &\rightarrow Type \ \mathbf{id} \\ Type &\rightarrow \mathbf{INT} \ | \ \mathbf{REAL} \ | \ \mathbf{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 \mathbf{BEGIN} \ Statement^* \ \mathbf{END} \\ Statement &\rightarrow Block \\ &\quad | \ LocalVarDecl \\ &\quad | \ AssignStmt \\ &\quad | \ ReturnStmt \\ &\quad | \ IfStmt \\ &\quad | \ WhileStmt \\ &\quad | \ WriteStmt \\ &\quad | \ ReadStmt \\ LocalVarDecl &\rightarrow Type \ AssignExpr \ (\ ',' \ AssignExpr \)^* \ ';' \\ AssignStmt &\rightarrow AssignExpr \ ';' \\ ReturnStmt &\rightarrow \mathbf{RETURN} \ BoolMultiExpr \ ';' \\ IfStmt &\rightarrow \mathbf{IF} \ ' (' \ BoolMultiExpr \)' \ Statement \ [\mathbf{ELSE} \ Statement] \\ WhileStmt &\rightarrow \mathbf{WHILE} \ ' (' \ BoolMultiExpr \)' \ Statement \\ WriteStmt &\rightarrow \mathbf{WRITE} \ ' (' \ BoolMultiExpr \ ',' \ \mathbf{string} \)' \ ';' \\ ReadStmt &\rightarrow \mathbf{READ} \ ' (' \ \mathbf{id} \ ',' \ \mathbf{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 \ (\ ('+' \ | \ '-') \ MultiplicativeExpr \)^* \\ MultiplicativeExpr &\rightarrow PrimaryExpr \ (\ ('*' \ | \ '/' \ | \ '%') \ PrimaryExpr \)^* \\ PrimaryExpr &\rightarrow \mathbf{num} \\ &\quad | \ \mathbf{id} \\ &\quad | \ \mathbf{TRUE} \\ &\quad | \ \mathbf{FALSE} \\ &\quad | \ '(' \ BoolMultiExpr \ ')' \\ &\quad | \ \mathbf{id} \ '(' \ [\ ActualParams \] \ ')' \\ AssignExpr &\rightarrow \mathbf{id} \ [\ ':' \ '=' \ BoolMultiExpr \] \\ BoolMultiExpr &\rightarrow BoolExpression \ (\ ('\mathbf{AND}' \ | \ '\mathbf{OR}') \ BoolExpression \)^* \\ BoolExpression &\rightarrow Expression \ [\ ('\mathbf{!}=' \ | \ '\mathbf{==}' \ | \ '\mathbf{>}' \ | \ '\mathbf{<}' \ | \ '\mathbf{>}' \ | \ '\mathbf{<}') \ Expression \] \\ &\quad | \ '\mathbf{!}' \ 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：

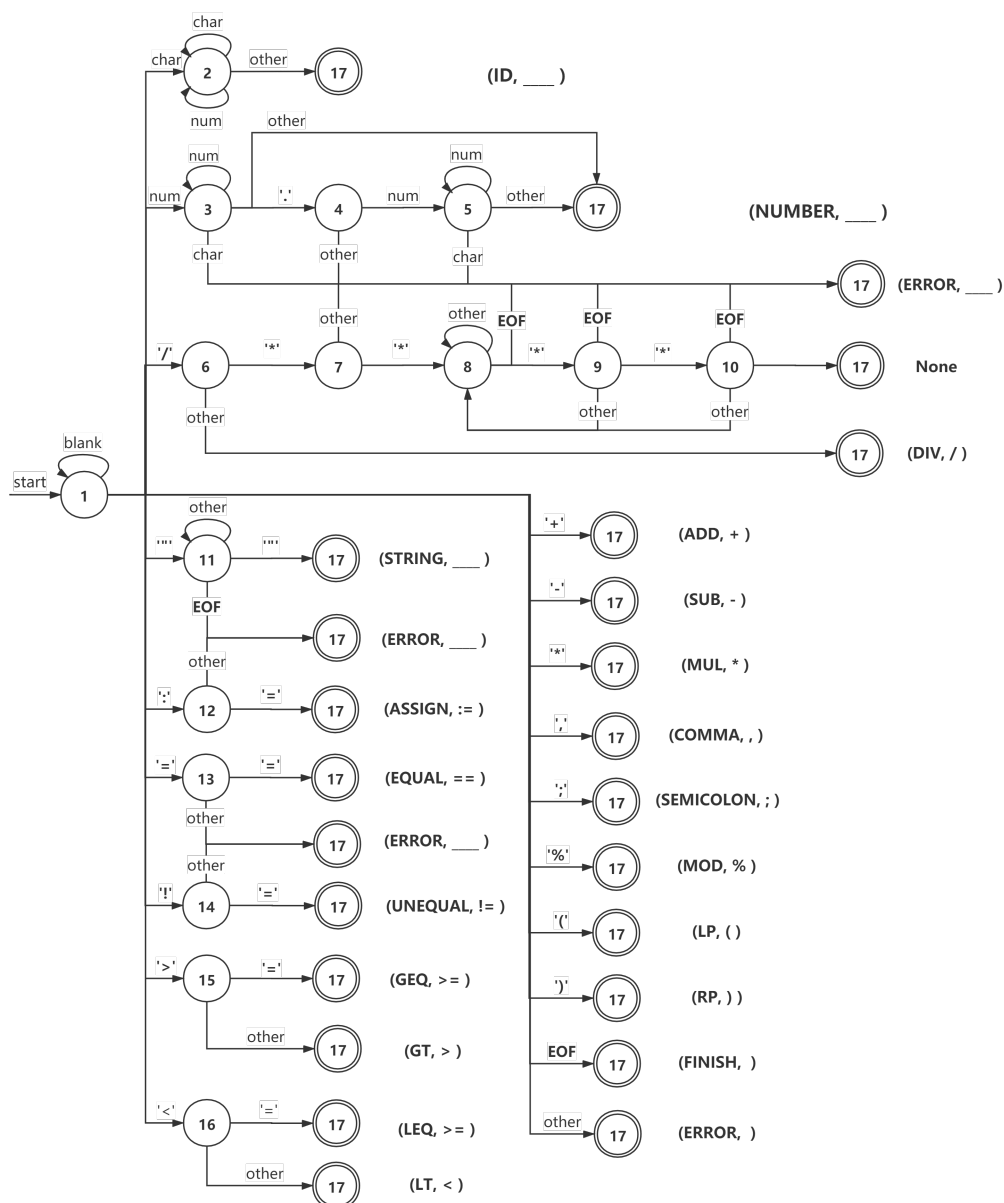


图 1: DFA 状态图

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：

```
1 typedef enum TokenType
2 {
3     // 关键字
4     IF,
5     ELSE,
6     WRITE,
7     READ,
8     RETURN,
9     BEGIN,
10    END,
11    MAIN,
12    INT,
13    REAL,
14    AND,
15    OR,
16    WHILE,
17    BOOL,
18    TRUE,
19    FALSE,
```

```

20 SEMICOLON, // ;
21 COMMA,    // ,
22 LP,       // (
23 RP,       // )
24 ADD,      // +
25 SUB,      // -
26 MUL,      // *
27 DIV,      // /
28 MOD,      // %
29 ASSIGN,   // :=
30 LEQ,      // <=
31 GEQ,      // >=
32 LT,       // <
33 GT,       // >
34 EQUAL,    // ==
35 UNEQUAL,  // !=
36 NOT,      // !
37 NUMBER,   // 数字
38 ID,       // 描述符
39 STRING,   // 字符串
40 // 特殊type
41 ERROR,
42 FINISH
43 } TokenType;

```

类似的，我同样按照4.3所述定义了 DFA 状态：

```

1 typedef enum DFAStatus
2 {
3     DFA_START,
4     .....
5     DFA_DONE
6 } DFAStatus;

```

TOKEN 结构如下所示：

```

1 typedef struct token
2 {
3     enum TokenType type; // token类型
4     char* str;           // token内容
5     int RowIndex;        // 起始行号
6     int ColIndex;        // 起始列号
7 } 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：

```

1 // 获取下一个TOKEN

```

```

2 struct token* getNextToken(FILE* source, int* RowIndexPtr, int* ColIndexPtr) {
3     // 创建并初始化TOKEN
4     struct token* tp = (struct token*)calloc(1, sizeof(struct token));
5     tp->str = (char*)calloc(MAXBUFLen, sizeof(char));
6     int StrIndex = 0;
7     int StrLen = MAXBUFLen;
8
9     char ch; // 当前字符
10    // 初始化DFA
11    enum DFAStatus CurrentStatus = DFA_START;
12    Bool isSave; // 是否存储该字符
13    Bool isRecordLocation = False; // 是否在TOKEN中记录位置
14    Bool isUpdateLocation = True; // 是否更新RowIndexPtr和ColIndexPtr存储的位置
15
16    while (True) {
17        // 获取下一个字符
18        ch = fgetc(source);
19        // 默认存储该字符
20        isSave = True;
21        // DFA状态跳转
22        switch (CurrentStatus) {
23            case DFA_START:
24                if (isBlankChar(ch) != True)
25                    isRecordLocation = True;
26                if (isBlankChar(ch) == True)
27                    isSave = False;
28                else if (isDigit(ch) == True)
29                    CurrentStatus = DFA_INTNUM;
30                else if (isAlpha(ch) == True)
31                    CurrentStatus = DFA_ID;
32                else if (ch == '"')
33                    CurrentStatus = DFA_STRING;
34                else if (ch == '/')
35                    CurrentStatus = DFA_DIV_OR_COMMENT;
36                else if (ch == ':')
37                    CurrentStatus = DFA_ASSIGN;
38                else if (ch == '=')
39                    CurrentStatus = DFA_EQUAL;
40                else if (ch == '!')
41                    CurrentStatus = DFA_NOT;
42                else if (ch == '>')
43                    CurrentStatus = DFA_GT;
44                else if (ch == '<')
45                    CurrentStatus = DFA_LT;
46                else {
47                    CurrentStatus = DFA_DONE;
48                    if (ch == ',')
49                        tp->type = COMMA;

```

```

50         else if (ch == ';')
51             tp->type = SEMICOLON;
52         .....
53         else if (ch == EOF) {
54             tp->type = FINISH;
55             isSave = False;
56         }
57         else
58             tp->type = ERROR;
59     }
60     break;
61     .....
62     default:
63         CurrentStatus = DFA_DONE;
64         tp->type = ERROR;
65         break;
66 }
67 // 存储TOKEN起始位置
68 if (isRecordLocation) {
69     tp->ColIndex = *ColIndexPtr;
70     tp->RowIndex = *RowIndexPtr;
71     isRecordLocation = False;
72 }
73 // 更新计数器记录的位置
74 if (isUpdateLocation) {
75     if (ch == '\n')
76     {
77         (*RowIndexPtr) += 1;
78         (*ColIndexPtr) = 1;
79     }
80     else
81         (*ColIndexPtr) += 1;
82 }
83 // 存储字符
84 if (isSave) {
85     // 空间不足，开辟新空间
86     if (StrIndex >= StrLen)
87     {
88         StrLen += MAXBUFLen;
89         tp->str = (char*)realloc(tp->str, StrLen * sizeof(char));
90     }
91     tp->str[StrIndex++] = ch;
92 }
93 // DFA到达结束状态，结束循环
94 if (CurrentStatus == DFA_DONE) {
95     // 存'\0'
96     if (StrIndex >= StrLen) {
97         StrLen += 1;

```

```

98         tp->str = (char*)realloc(tp->str, StrLen * sizeof(char));
99     }
100    tp->str[StrIndex++] = '\0';
101    // 若为错误, 报错
102    if (tp->type == ERROR)
103        printf("\n[ERROR] %d:%d: %s\n", tp->RowIndex, tp->ColIndex, tp->str);
104    // 将特定ID转换为关键字
105    else if (tp->type == ID) {
106        for (int i = 0; i < KeyTokensLen; i++) {
107            if (strcmp(tp->str, KeyTokens[i].str) == 0) {
108                tp->type = KeyTokens[i].type;
109                break;
110            }
111        }
112    }
113    break;
114 }
115 }
116 return tp;
117 }

```

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:

```

1 // 节点
2 typedef struct TreeNode
3 {
4     NodeType nodetype; // 节点类型
5     struct TreeNode** ChildPtrList; // 子节点指针数组的指针
6     int ChildCurrentIndex; // 当前子节点数量
7     int ChildMaxNum; // 子节点指针数组支持的最大数量
8     token* tp; // 叶子节点对应的TOKEN
9 } 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。

```

1 // 节点类型
2 typedef enum NodeType
3 {
4     NT_PROGRAM,
5     NT_FUNCTION_DECL,
6     NT_TYPE,
7     NT_FORMAL_PARAMS,
8     NT_FORMAL_PARAM,

```

```

9     NT_BLOCK,
10    NT_STATEMENT,
11    NT_LOCAL_VAR_DECL,
12    NT_ASSIGN_STMT,
13    NT_RETURN_STMT,
14    NT_IF_STMT,
15    NT_WHILE_STMT,
16    NT_WRITE_STMT,
17    NT_READ_STMT,
18    NT_EXPRESSION,
19    NT_MULTIPLICATIVE_EXPR,
20    NT_PRIMARY_EXPR,
21    NT_ASSIGN_EXPR,
22    NT_BOOL_MULTI_EXPR,
23    NT_BOOL_EXPRESSION,
24    NT_ACTUAL_PARAMS,
25    NT_FACTOR,
26 } NodeType;

```

各个非终结符号对应的过程函数列表如下所示：

```

1  TreeNode* match(token** tpl, int* IndexPtr, int MaxTokenNum, TokenType type, Bool isThrow)
   ;
2  TreeNode* analyseProgram(token** tpl, int MaxTokenNum);
3  TreeNode* analyseFunctionDecl(token** tpl, int* IndexPtr, int MaxTokenNum);
4  TreeNode* analyseType(token** tpl, int* IndexPtr, int MaxTokenNum);
5  TreeNode* analyseFormalParams(token** tpl, int* IndexPtr, int MaxTokenNum);
6  TreeNode* analyseFormalParam(token** tpl, int* IndexPtr, int MaxTokenNum);
7  Bool isFormalParam(token** tpl, int* IndexPtr, int MaxTokenNum);
8  TreeNode* analyseBlock(token** tpl, int* IndexPtr, int MaxTokenNum);
9  TreeNode* analyseStatement(token** tpl, int* IndexPtr, int MaxTokenNum);
10 TreeNode* analyseLocalVarDecl(token** tpl, int* IndexPtr, int MaxTokenNum);
11 TreeNode* analyseAssignStmt(token** tpl, int* IndexPtr, int MaxTokenNum);
12 TreeNode* analyseReturnStmt(token** tpl, int* IndexPtr, int MaxTokenNum);
13 TreeNode* analyseIfStmt(token** tpl, int* IndexPtr, int MaxTokenNum);
14 TreeNode* analyseWhileStmt(token** tpl, int* IndexPtr, int MaxTokenNum);
15 TreeNode* analyseWriteStmt(token** tpl, int* IndexPtr, int MaxTokenNum);
16 TreeNode* analyseReadStmt(token** tpl, int* IndexPtr, int MaxTokenNum);
17 TreeNode* analyseExpression(token** tpl, int* IndexPtr, int MaxTokenNum);
18 TreeNode* analyseMultiplicativeExpr(token** tpl, int* IndexPtr, int MaxTokenNum);
19 TreeNode* analysePrimaryExpr(token** tpl, int* IndexPtr, int MaxTokenNum);
20 TreeNode* analyseAssignExpr(token** tpl, int* IndexPtr, int MaxTokenNum);
21 TreeNode* analyseBoolMultiExpr(token** tpl, int* IndexPtr, int MaxTokenNum);
22 TreeNode* analyseBoolExpression(token** tpl, int* IndexPtr, int MaxTokenNum);
23 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：

```
1  TreeNode* analyseFunctionDecl(token** tpl, int* IndexPtr, int MaxTokenNum) {
2      // 创建新节点
3      TreeNode* fp = buildTreeNode(False);
4      int ChildIndex = 0;
5      fp->nodetype = NT_FUNCTION_DECL;
6      TreeNode* cp = NULL;
7      // 尝试匹配非终结符号Type，若失败则报错返回空指针
8      cp = analyseType(tpl, IndexPtr, MaxTokenNum);
9      if (cp != NULL)
10         addChildNode(fp, cp);
11     else {
12         if (*IndexPtr < MaxTokenNum)
13             throwParserError(tpl[*IndexPtr], "Type doesn't finish.");
14         else
15             throwParserError(tpl[*IndexPtr - 1], "Type doesn't finish.");
16         return NULL;
17     }
18     // 尝试匹配终结符号MAIN，允许失败
19     cp = match(tpl, IndexPtr, MaxTokenNum, MAIN, False);
20     if (cp != NULL)
21         addChildNode(fp, cp);
22     // 尝试匹配终结符号id，若失败则报错返回空指针
23     cp = match(tpl, IndexPtr, MaxTokenNum, ID, True);
24     if (cp != NULL)
25         addChildNode(fp, cp);
26     else
27         return NULL;
28     // 尝试匹配终结符号(，若失败则报错返回空指针
29     cp = match(tpl, IndexPtr, MaxTokenNum, LP, True);
30     if (cp != NULL)
31         addChildNode(fp, cp);
32     else
33         return NULL;
34     // 尝试匹配非终结符号FormalParams，若失败则报错返回空指针
35     cp = analyseFormalParams(tpl, IndexPtr, MaxTokenNum);
36     if (cp != NULL)
37         addChildNode(fp, cp);
38     else {
39         if (*IndexPtr < MaxTokenNum)
40             throwParserError(tpl[*IndexPtr], "FormalParams doesn't finish.");
41         else
42             throwParserError(tpl[*IndexPtr - 1], "FormalParams doesn't finish.");
43         return NULL;
44     }
45     // 尝试匹配终结符号)，若失败则报错返回空指针
46     cp = match(tpl, IndexPtr, MaxTokenNum, RP, True);
47     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  {
3      TreeNode* np = NULL;
4      // 若匹配对应类型
5      if (*IndexPtr < MaxTokenNum && tpl[*IndexPtr]->type == type) {
6          // 创建新节点并赋值
7          np = buildTreeNode(True);
8          np->nodetype = NT_FACTOR;
9          np->tp = tpl[*IndexPtr];
10         // 偏移+1
11         (*IndexPtr)++;
12     }
13     // 不匹配，报错
14     if (isThrow && np == NULL) {
15         char output[MAXMESLEN];
16         if (*IndexPtr >= MaxTokenNum) {
17             snprintf(output, MAXMESLEN, "(%s,%s) Expect a %s.", getTokenName(tpl[*IndexPtr
18                 - 1]->type), tpl[*IndexPtr - 1]->str, getTokenName(type));
19             throwParserError(tpl[*IndexPtr - 1], output);
20         }
21         else {
22             snprintf(output, MAXMESLEN, "(%s,%s) Expect a %s.", getTokenName(tpl[*IndexPtr
23                 ]->type), tpl[*IndexPtr]->str, getTokenName(type));
24             throwParserError(tpl[*IndexPtr], output);
25         }
26     }
27     return np;
28 }

```

用于添加子节点的函数 addChildNode() 如下所示：


```

1 // 添加子节点
2 void addChildNode(TreeNode* fp, TreeNode* cp) {
3     // 空间不够的话扩容
4     if (fp->ChildCurrentIndex >= fp->ChildMaxNum) {
5         fp->ChildMaxNum += CHILDMAXNUM;
6         fp->ChildPtrList = (TreeNode**) realloc(fp->ChildPtrList, fp->ChildMaxNum * sizeof(
            TreeNode*));
7         for (int i = fp->ChildCurrentIndex; i < fp->ChildMaxNum; i++)
8             fp->ChildPtrList[i] = NULL;
9     }
10    // 添加
11    fp->ChildPtrList[fp->ChildCurrentIndex++] = cp;
12 }

```

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：

```

1 /** this is a comment line in the sample program */
2 INT f2(INT x, INT y )
3 BEGIN
4     INT z;
5     z := x*x - y*y;
6     RETURN z;
7 END
8 INT MAIN f1 ()
9 BEGIN
10    INT x;
11    READ(x, "A41.input");
12    INT y;
13    READ(y, "A42.input");
14    INT z;
15    z := f2(x,y) + f2(y,x);
16    WRITE (z, "A4.output");
17 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](#)。

```

1 (INT, INT)
2 (ID, f2)
3 (LP, ())
4 (INT, INT)
5 (ID, x)

```

```

6 (COMMA, ,)
7 (INT, INT)
8 (ID, y)
9 (RP, ))
10 (BEGIN, BEGIN)
11 (INT, INT)
12 (ID, z)
13 (SEMICOLON, ;)
14 .....
15 (WRITE, WRITE)
16 (LP, ())
17 (ID, z)
18 (COMMA, ,)
19 (STRING, "A4.output")
20 (RP, ))
21 (SEMICOLON, ;)
22 (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：

```

1  /** This is a TINY+ program without bugs. */
2  INT f2 (INT x, INT y )
3  BEGIN
4      INT a;
5      a := 10;
6      WHILE (a >= x)
7          a := a % 2;
8      RETURN a+y;
9  END
10 INT MAIN f1 ()
11 BEGIN
12     BOOL flag:=TRUE;
13     INT x:=5, y;
14     REAL c:=4.521;
15     IF (flag)
16         x := (x+3)*4;
17     ELSE
18     BEGIN
19         READ(y, "input y");
20         c := y - 3;
21     END
22     INT z:= f2(x,y) + f2(y,x);
23     WRITE (z, "output z");
24 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](#)。

```

1 (INT, INT)
2 (ID, f2)
3 (LP, ())
4 (INT, INT)
5 (ID, x)
6 (COMMA, ,)
7 (INT, INT)
8 (ID, y)
9 (RP, ))
10 (BEGIN, BEGIN)
11 (INT, INT)
12 (ID, a)
13 (SEMICOLON, ;)
14 (ID, a)
15 (ASSIGN, :=)
16 (NUMBER, 10)
17 (SEMICOLON, ;)
18 .....
19 (WRITE, WRITE)
20 (LP, ())
21 (ID, z)
22 (COMMA, ,)
23 (STRING, "output z")
24 (RP, ))
25 (SEMICOLON, ;)
26 (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：

```

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

```

1 [INFO] Lexical analysis:
2 (INT, INT)
3 (MAIN, MAIN)
4 (ID, f)
5 (LP, ())
6 (RP, ))

```

```

7 (BEGIN, BEGIN)
8 (BOOL, BOOL)
9 (ID, flag)
10 (ASSIGN, :=)
11 (TRUE, TRUE)
12 (SEMICOLON, ;)
13 (INT, INT)
14
15 [ERROR] 5:9: 3x
16
17 [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：

```

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

```

可见第 1 行代码中，“/**” 漏了一个 “*”。

词法分析器输出如下所示：

```

1 [INFO] Lexical analysis:
2
3 [ERROR] 1:1: /*
4
5 [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。

```

1 Program
2 |__ FuntionDecl
3 |   |__ Type
4 |   |   |__ Factor: (INT, INT)
5 |   |   |.....
6 |   |__ Block
7 |       |__ Factor: (BEGIN, BEGIN)
8 |       |__ Statement
9 |           |__ LocalVarDecl
10 |           |   |__ Type
11 |           |   |   |__ Factor: (INT, INT)
12 |           |   |   |__ AssignExpr
13 |           |   |   |   |__ Factor: (ID, z)
14 |           |   |   |   |__ Factor: (SEMICOLON, ;)
15 |           |   |   |.....
16 |           |   |__ Factor: (END, END)
17 |__ FuntionDecl
18 |   |__ Type
19 |   |   |__ Factor: (INT, INT)
20 |   |   |.....
21 |   |__ Block
22 |       |__ Factor: (BEGIN, BEGIN)
23 |       |.....
24 |       |__ Statement
25 |           |__ WriteStmt
26 |               |__ Factor: (WRITE, WRITE)
27 |               |__ Factor: (LP, ()
28 |               |__ BoolMultiExpr
29 |                   |__ BoolExpression
30 |                       |__ Expression
31 |                           |__ MultiplicativeExpr
32 |                               |__ PrimaryExpr
33 |                                   |__ Factor: (ID, z)
34 |                                   |__ Factor: (COMMA, ,)
35 |                                   |__ Factor: (STRING, "A4.output")
36 |                                   |__ Factor: (RP, ))
37 |                                   |__ Factor: (SEMICOLON, ;)
38 |                                   |__ Factor: (END, END)

```

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。

```

1 Program
2 |__ FuntionDecl
3 |   |__ Type
4 |   |   |__ Factor: (INT, INT)
5 |   |__ Factor: (ID, f2)
6 |   |__ Factor: (LP, ())
7 |   |.....
8 |   |__ Factor: (RP, ))
9 |   |__ Block
10 |       |__ Factor: (BEGIN, BEGIN)
11 |       |.....
12 |       |__ Statement
13 |           |__ ReturnStmt
14 |           |   |__ Factor: (RETURN, RETURN)
15 |           |__ BoolMultiExpr
16 |           |   |__ BoolExpression
17 |           |       |__ Expression
18 |           |           |__ MultiplicativeExpr
19 |           |           |   |__ PrimaryExpr
20 |           |           |       |__ Factor: (ID, a)
21 |           |           |       |__ Factor: (ADD, +)
22 |           |           |       |__ MultiplicativeExpr
23 |           |           |       |__ PrimaryExpr
24 |           |           |       |__ Factor: (ID, y)
25 |           |       |__ Factor: (SEMICOLON, ;)
26 |       |__ Factor: (END, END)
27 |__ FuntionDecl
28 |.....
29 |__ Block
30 |   |__ Factor: (BEGIN, BEGIN)
31 |   |.....
32 |   |__ Statement
33 |       |__ WriteStmt
34 |       |   |__ Factor: (WRITE, WRITE)
35 |       |   |__ Factor: (LP, ())
36 |       |   |__ BoolMultiExpr
37 |       |       |__ BoolExpression
38 |       |       |__ Expression
39 |       |           |__ MultiplicativeExpr
40 |       |           |   |__ PrimaryExpr
41 |       |           |       |__ Factor: (ID, z)
42 |       |       |__ Factor: (COMMA, ,)
43 |       |       |__ Factor: (STRING, "output z")
44 |       |       |__ Factor: (RP, ))
45 |       |       |__ Factor: (SEMICOLON, ;)
46 |   |__ Factor: (END, END)
47

```

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
```

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

```

1 | TINY+ v0.4 :: 18308013 ChenJiahao |
2 |
3 |
4 [INFO] Welcome to use TINY+ compiler!
5 [INFO] Please input your source file full path:
6 [INPUT] E:\code\Windows\C++\Compiler_Principle\ex3\my_compiler\good_sample.tiny
7 [INFO] Lexical analysis:
8
9 (INT, INT)
10 (ID, f2)
11 (LP, ())
12 (INT, INT)
13 (ID, x)
14 (COMMA, ,)
15 (INT, INT)
16 (ID, y)
17 (RP, ))
18 (BEGIN, BEGIN)
19 (INT, INT)
20 (ID, z)
21 (SEMICOLON, ;)
22 (ID, z)
23 (ASSIGN, :=)

```



```

24 (ID, x)
25 (MUL, *)
26 (ID, x)
27 (SUB, -)
28 (ID, y)
29 (MUL, *)
30 (ID, y)
31 (SEMICOLON, ;)
32 (RETURN, RETURN)
33 (ID, z)
34 (SEMICOLON, ;)
35 (END, END)
36 (INT, INT)
37 (MAIN, MAIN)
38 (ID, f1)
39 (LP, ())
40 (RP, ))
41 (BEGIN, BEGIN)
42 (INT, INT)
43 (ID, x)
44 (SEMICOLON, ;)
45 (READ, READ)
46 (LP, ())
47 (ID, x)
48 (COMMA, ,)
49 (STRING, "A41.input")
50 (RP, ))
51 (SEMICOLON, ;)
52 (INT, INT)
53 (ID, y)
54 (SEMICOLON, ;)
55 (READ, READ)
56 (LP, ())
57 (ID, y)
58 (COMMA, ,)
59 (STRING, "A42.input")
60 (RP, ))
61 (SEMICOLON, ;)
62 (INT, INT)
63 (ID, z)
64 (SEMICOLON, ;)
65 (ID, z)
66 (ASSIGN, :=)
67 (ID, f2)
68 (LP, ())
69 (ID, x)
70 (COMMA, ,)
71 (ID, y)

```

```

72 (RP, ))
73 (ADD, +)
74 (ID, f2)
75 (LP, ()
76 (ID, y)
77 (COMMA, ,)
78 (ID, x)
79 (RP, ))
80 (SEMICOLON, ;)
81 (WRITE, WRITE)
82 (LP, ()
83 (ID, z)
84 (COMMA, ,)
85 (STRING, "A4.output")
86 (RP, ))
87 (SEMICOLON, ;)
88 (END, END)
89
90 [INFO] Lexical analysis Finished.
91 [INFO] Parser analysis:
92
93 Program
94 |__ FuntionDecl
95 |   |__ Type
96 |   |   |__ Factor: (INT, INT)
97 |   |   |__ Factor: (ID, f2)
98 |   |   |__ Factor: (LP, ()
99 |   |   |__ FormalParams
100 |   |   |   |__ FormalParam
101 |   |   |   |   |__ Type
102 |   |   |   |   |__ Factor: (INT, INT)
103 |   |   |   |   |__ Factor: (ID, x)
104 |   |   |   |   |__ Factor: (COMMA, ,)
105 |   |   |   |   |__ FormalParam
106 |   |   |   |   |__ Type
107 |   |   |   |   |__ Factor: (INT, INT)
108 |   |   |   |   |__ Factor: (ID, y)
109 |   |   |__ Factor: (RP, ))
110 |   |__ Block
111 |       |__ Factor: (BEGIN, BEGIN)
112 |       |__ Statement
113 |       |   |__ LocalVarDecl
114 |       |       |__ Type
115 |       |       |   |__ Factor: (INT, INT)
116 |       |       |   |__ AssignExpr
117 |       |       |   |__ Factor: (ID, z)
118 |       |       |   |__ Factor: (SEMICOLON, ;)
119 |       |__ 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
217						__ Factor: (ID, x)
218					__ Factor:	(COMMA, ,)
219					__ Expression	
220					__ MultiplicativeExpr	
221					__ PrimaryExpr	
222					__ Factor: (ID, y)	
223				__ Factor:	(RP,))	
224			__ Factor:	(ADD, +)		
225			__ MultiplicativeExpr			
226			__ PrimaryExpr			
227			__ Factor:	(ID, f2)		
228			__ Factor:	(LP, ())		
229			__ ActualParams			
230			__ Expression			
231			__ MultiplicativeExpr			
232			__ PrimaryExpr			
233			__ Factor:	(ID, y)		
234			__ Factor:	(COMMA, ,)		
235			__ Expression			
236			__ MultiplicativeExpr			
237			__ PrimaryExpr			
238			__ Factor:	(ID, x)		
239			__ Factor:	(RP,))		
240		__ Factor:	(SEMICOLON, ;)			
241	__ Statement					
242	__ WriteStmt					
243	__ Factor:	(WRITE, WRITE)				
244	__ Factor:	(LP, ())				
245	__ BoolMultiExpr					
246	__ BoolExpression					
247	__ Expression					
248	__ MultiplicativeExpr					
249	__ PrimaryExpr					
250	__ Factor:	(ID, z)				
251	__ Factor:	(COMMA, ,)				
252	__ Factor:	(STRING, "A4.output")				
253	__ Factor:	(RP,))				
254	__ Factor:	(SEMICOLON, ;)				
255	__ Factor:	(END, END)				
256						
257	[INFO]	Parser analysis Finished.				

8.2 good_example.tinyplus 完整分析输出

1

2

TINY+ v0.4 :: 18308013 ChenJiahao

```

3
4 [INFO] Welcome to use TINY+ compiler!
5 [INFO] Please input your source file full path:
6 [INPUT] E:\code\Windows\C++\Compiler_Principle\ex3\my_compiler\good_sample.tinyplus
7 [INFO] Lexical analysis:
8
9 (INT, INT)
10 (ID, f2)
11 (LP, ())
12 (INT, INT)
13 (ID, x)
14 (COMMA, ,)
15 (INT, INT)
16 (ID, y)
17 (RP, ))
18 (BEGIN, BEGIN)
19 (INT, INT)
20 (ID, a)
21 (SEMICOLON, ;)
22 (ID, a)
23 (SEMICOLON, ;)
24 (ID, a)
25 (ASSIGN, :=)
26 (NUMBER, 10)
27 (SEMICOLON, ;)
28 (WHILE, WHILE)
29 (LP, ())
30 (ID, a)
31 (GEQ, >=)
32 (ID, x)
33 (RP, ))
34 (ID, a)
35 (ASSIGN, :=)
36 (ID, a)
37 (MOD, %)
38 (NUMBER, 2)
39 (SEMICOLON, ;)
40 (RETURN, RETURN)
41 (ID, a)
42 (ADD, +)
43 (ID, y)
44 (SEMICOLON, ;)
45 (END, END)
46 (INT, INT)
47 (MAIN, MAIN)
48 (ID, f1)
49 (LP, ())
50 (RP, ))

```

```

51 (BEGIN, BEGIN)
52 (BOOL, BOOL)
53 (ID, flag)
54 (ASSIGN, :=)
55 (TRUE, TRUE)
56 (SEMICOLON, ;)
57 (INT, INT)
58 (ID, x)
59 (ASSIGN, :=)
60 (NUMBER, 5)
61 (COMMA, ,)
62 (ID, y)
63 (SEMICOLON, ;)
64 (REAL, REAL)
65 (ID, c)
66 (ASSIGN, :=)
67 (NUMBER, 4.521)
68 (SEMICOLON, ;)
69 (IF, IF)
70 (LP, ())
71 (ID, flag)
72 (RP, ))
73 (ID, x)
74 (ASSIGN, :=)
75 (LP, ())
76 (ID, x)
77 (ADD, +)
78 (NUMBER, 3)
79 (RP, ))
80 (MUL, *)
81 (NUMBER, 4)
82 (SEMICOLON, ;)
83 (ELSE, ELSE)
84 (BEGIN, BEGIN)
85 (READ, READ)
86 (LP, ())
87 (ID, y)
88 (COMMA, ,)
89 (STRING, "input y")
90 (RP, ))
91 (SEMICOLON, ;)
92 (ID, c)
93 (ASSIGN, :=)
94 (ID, y)
95 (SUB, -)
96 (NUMBER, 3)
97 (SEMICOLON, ;)
98 (END, END)

```

```

99 (INT, INT)
100 (ID, z)
101 (ASSIGN, :=)
102 (ID, f2)
103 (LP, ())
104 (ID, x)
105 (COMMA, ,)
106 (ID, y)
107 (RP, ))
108 (ADD, +)
109 (ID, f2)
110 (LP, ())
111 (ID, y)
112 (COMMA, ,)
113 (ID, x)
114 (RP, ))
115 (SEMICOLON, ;)
116 (WRITE, WRITE)
117 (LP, ())
118 (ID, z)
119 (COMMA, ,)
120 (STRING, "output z")
121 (RP, ))
122 (SEMICOLON, ;)
123 (END, END)
124
125 [INFO] Lexical analysis Finished.
126 [INFO] Parser analysis:
127
128 Program
129 |__ FuntionDecl
130 |   |__ Type
131 |   |   |__ Factor: (INT, INT)
132 |   |   |__ Factor: (ID, f2)
133 |   |   |__ Factor: (LP, ())
134 |   |   |__ FormalParams
135 |   |   |   |__ FormalParam
136 |   |   |   |   |__ Type
137 |   |   |   |   |   |__ Factor: (INT, INT)
138 |   |   |   |   |   |__ Factor: (ID, x)
139 |   |   |   |   |   |__ Factor: (COMMA, ,)
140 |   |   |   |   |   |__ FormalParam
141 |   |   |   |   |   |   |__ Type
142 |   |   |   |   |   |   |   |__ Factor: (INT, INT)
143 |   |   |   |   |   |   |   |__ Factor: (ID, y)
144 |   |   |   |   |   |   |   |__ Factor: (RP, ))
145 |   |   |   |   |   |   |   |__ Block
146 |   |   |   |   |   |   |   |   |__ Factor: (BEGIN, BEGIN)

```



```

147 | | |__ Statement
148 | | |__ LocalVarDecl
149 | | |__ Type
150 | | |__ Factor: (INT, INT)
151 | | |__ AssignExpr
152 | | |__ Factor: (ID, a)
153 | | |__ Factor: (SEMICOLON, ;)
154 | |__ Statement
155 | |__ AssignStmt
156 | |__ AssignExpr
157 | |__ Factor: (ID, a)
158 | |__ Factor: (SEMICOLON, ;)
159 | |__ Statement
160 | |__ AssignStmt
161 | |__ AssignExpr
162 | |__ Factor: (ID, a)
163 | |__ Factor: (ASSIGN, :=)
164 | |__ BoolMultiExpr
165 | |__ BoolExpression
166 | |__ Expression
167 | |__ MultiplicativeExpr
168 | |__ PrimaryExpr
169 | |__ Factor: (NUMBER, 10)
170 | |__ Factor: (SEMICOLON, ;)
171 | |__ Statement
172 | |__ WhileStmt
173 | |__ Factor: (WHILE, WHILE)
174 | |__ Factor: (LP, ())
175 | |__ BoolMultiExpr
176 | |__ BoolExpression
177 | |__ Expression
178 | |__ MultiplicativeExpr
179 | |__ PrimaryExpr
180 | |__ Factor: (ID, a)
181 | |__ Factor: (GEQ, >=)
182 | |__ Expression
183 | |__ MultiplicativeExpr
184 | |__ PrimaryExpr
185 | |__ Factor: (ID, x)
186 | |__ Factor: (RP, ))
187 | |__ Statement
188 | |__ AssignStmt
189 | |__ AssignExpr
190 | |__ Factor: (ID, a)
191 | |__ Factor: (ASSIGN, :=)
192 | |__ BoolMultiExpr
193 | |__ BoolExpression
194 | |__ Expression

```

```

195 |         |         |         |___ MultiplicativeExpr
196 |         |         |         |___ PrimaryExpr
197 |         |         |         |___ Factor: (ID, a)
198 |         |         |         |___ Factor: (MOD, %)
199 |         |         |         |___ PrimaryExpr
200 |         |         |         |___ Factor: (NUMBER, 2)
201 |         |         |___ Factor: (SEMICOLON, ;)
202 |         |___ Statement
203 |         |___ ReturnStmt
204 |         |___ Factor: (RETURN, RETURN)
205 |         |___ BoolMultiExpr
206 |         |___ BoolExpression
207 |         |___ Expression
208 |         |         |___ MultiplicativeExpr
209 |         |         |___ PrimaryExpr
210 |         |         |___ Factor: (ID, a)
211 |         |         |___ Factor: (ADD, +)
212 |         |         |___ MultiplicativeExpr
213 |         |         |___ PrimaryExpr
214 |         |         |___ Factor: (ID, y)
215 |         |___ Factor: (SEMICOLON, ;)
216 |         |___ Factor: (END, END)
217 |___ FuntionDecl
218 |___ Type
219 |___ Factor: (INT, INT)
220 |___ Factor: (MAIN, MAIN)
221 |___ Factor: (ID, f1)
222 |___ Factor: (LP, ())
223 |___ FormalParams
224 |___ Factor: (RP, ))
225 |___ Block
226 |___ Factor: (BEGIN, BEGIN)
227 |___ Statement
228 |___ LocalVarDecl
229 |___ Type
230 |___ Factor: (BOOL, BOOL)
231 |___ AssignExpr
232 |___ Factor: (ID, flag)
233 |___ Factor: (ASSIGN, :=)
234 |___ BoolMultiExpr
235 |___ BoolExpression
236 |___ Expression
237 |___ MultiplicativeExpr
238 |___ PrimaryExpr
239 |___ Factor: (TRUE, TRUE)
240 |___ Factor: (SEMICOLON, ;)
241 |___ Statement
242 |___ 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

```

```

291 |                                     |__ MultiplicativeExpr
292 |                                     |__ PrimaryExpr
293 |                                     |   |__ Factor: (LP, ())
294 |                                     |   |__ BoolMultiExpr
295 |                                     |       |__ BoolExpression
296 |                                     |       |__ Expression
297 |                                     |       |__ MultiplicativeExpr
298 |                                     |           |__ PrimaryExpr
299 |                                     |           |__ Factor: (ID, x)
300 |                                     |           |__ Factor: (ADD, +)
301 |                                     |           |__ MultiplicativeExpr
302 |                                     |           |__ PrimaryExpr
303 |                                     |           |__ Factor: (NUMBER,
    3)
304 |                                     |   |__ Factor: (RP, ))
305 |                                     |__ Factor: (MUL, *)
306 |                                     |__ PrimaryExpr
307 |                                     |__ Factor: (NUMBER, 4)
308 |                                     |__ Factor: (SEMICOLON, ;)
309 |                               |__ Factor: (ELSE, ELSE)
310 |                               |__ Statement
311 |                                 |__ Block
312 |                                 |__ Factor: (BEGIN, BEGIN)
313 |                                 |__ Statement
314 |                                   |__ ReadStmt
315 |                                   |__ Factor: (READ, READ)
316 |                                   |__ Factor: (LP, ())
317 |                                   |__ Factor: (ID, y)
318 |                                   |__ Factor: (COMMA, ,)
319 |                                   |__ Factor: (STRING, "input y")
320 |                                   |__ Factor: (RP, ))
321 |                                   |__ Factor: (SEMICOLON, ;)
322 |                               |__ Statement
323 |                               |   |__ AssignStmt
324 |                               |       |__ AssignExpr
325 |                               |           |__ Factor: (ID, c)
326 |                               |           |__ Factor: (ASSIGN, :=)
327 |                               |           |__ BoolMultiExpr
328 |                               |               |__ BoolExpression
329 |                               |               |__ Expression
330 |                               |               |__ MultiplicativeExpr
331 |                               |                   |__ PrimaryExpr
332 |                               |                   |__ Factor: (ID, y)
333 |                               |                   |__ Factor: (SUB, -)
334 |                               |                   |__ MultiplicativeExpr
335 |                               |                       |__ PrimaryExpr
336 |                               |                       |__ Factor: (NUMBER, 3)
337 |                               |__ Factor: (SEMICOLON, ;)

```

```

338 |         |__ Factor: (END, END)
339 |__ Statement
340 |   |__ LocalVarDecl
341 |       |__ Type
342 |       |   |__ Factor: (INT, INT)
343 |       |__ AssignExpr
344 |       |   |__ Factor: (ID, z)
345 |       |   |__ Factor: (ASSIGN, :=)
346 |       |   |__ BoolMultiExpr
347 |       |       |__ BoolExpression
348 |       |       |__ Expression
349 |       |           |__ MultiplicativeExpr
350 |       |           |   |__ PrimaryExpr
351 |       |           |       |__ Factor: (ID, f2)
352 |       |           |       |__ Factor: (LP, ()
353 |       |           |       |__ ActualParams
354 |       |           |       |   |__ Expression
355 |       |           |       |       |__ MultiplicativeExpr
356 |       |           |       |       |   |__ PrimaryExpr
357 |       |           |       |       |       |__ Factor: (ID, x)
358 |       |           |       |       |   |__ Factor: (COMMA, ,)
359 |       |           |       |       |   |__ Expression
360 |       |           |       |       |   |__ MultiplicativeExpr
361 |       |           |       |       |       |__ PrimaryExpr
362 |       |           |       |       |       |__ Factor: (ID, y)
363 |       |           |       |       |   |__ Factor: (RP, ))
364 |       |           |   |__ Factor: (ADD, +)
365 |       |           |__ MultiplicativeExpr
366 |       |           |   |__ PrimaryExpr
367 |       |           |       |__ Factor: (ID, f2)
368 |       |           |       |__ Factor: (LP, ()
369 |       |           |       |__ ActualParams
370 |       |           |       |   |__ Expression
371 |       |           |       |       |__ MultiplicativeExpr
372 |       |           |       |       |   |__ PrimaryExpr
373 |       |           |       |       |       |__ Factor: (ID, y)
374 |       |           |       |       |   |__ Factor: (COMMA, ,)
375 |       |           |       |       |   |__ Expression
376 |       |           |       |       |   |__ MultiplicativeExpr
377 |       |           |       |       |       |__ PrimaryExpr
378 |       |           |       |       |       |__ Factor: (ID, x)
379 |       |           |       |   |__ Factor: (RP, ))
380 |       |   |__ Factor: (SEMICOLON, ;)
381 |__ Statement
382 |   |__ WriteStmt
383 |       |__ Factor: (WRITE, WRITE)
384 |       |__ Factor: (LP, ()
385 |       |__ 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.

```