The report of PL0 Compiler

—— on *Compilation Principles*

摘 要

本实验报告的编写目的是为了全面地说明和分析本学期《编译原理与技术》实验的情况成果。其将从设计的目的、环境、原则开始介绍，开始设计前明确的原则是整个实验的可读性、可维护性、逻辑性的保证。课程设计一共分 5 个子实验，其中实验一是词法分析器设计，主要是设计词法基本单位的枚举承载类和符号类等，这是后续实验的基础；实验二是语法分析器的设计，采用递归下降分析法，将各个规约项分而治之地处理；实验三是语法树生成，由于语法分析器设计得解耦性好，本实验提供一种入侵程度低的修改方案，使得语法树能方便生成；实验四是目标代码生成，因为语法分析和目标代码生成是在一遍扫描源代码时同时完成，所以目标代码生成部分的实现对语法分析部分有强侵入性，本实验采用编码规范强制要求加注释以区分二者；实验五是目标代码的解释执行，设计一个与语法分析器分离的解释执行器，单独对生成的目标代码表进行模拟执行。附录将附带一些具体的测试样例和各个子模块的测试结果。在该实验报告中，您将看到我在解决各个子实验时的方案，设计过程中的思路，相关编译原理的总结和面向对象设计模式的使用，以及最重要的我对整个编译原理课程的思考和收获。

关键词：PL0、编译器、仿真、递归下降分析法、面向对象

目录

1 前言 1

1.1 设计目的 1

1.2 设计环境 1

1.3 设计原则 1

1.4 PL/0 语言文法的 BNF 表示 1

1.5 PL/0 编译系统结构 2

2 词法分析器 3

2.1 任务要求 3

2.2 符号类型类的设计 3

2.3 符号类的设计 5

2.4 词法分析器类的设计 7

2.5 测试代码 9

3 语法分析器 11

3.1 任务要求 11

3.2 异常处理器的设计 11

3.3 符号表及其管理器的设计 13

3.4 语法分析器的设计 16

3.5 测试代码 25

4 语法树生成 26

4.1 任务要求 26

4.2 语法树类的设计 26

4.3 语法分析器改动部分 28

4.4 测试代码 29

5 目标代码生成 30

5.1 任务要求 30

5.2 目标代码类的设计 30

5.3 语法分析器的改动部分 32

5.4 测试代码 34

6 解释执行器 35

6.1 任务要求 35
6.2 执行器的设计 35
6.3 测试代码 40
7 附录 41
7.1 测试样例代码 41
7.1.1 PL0_code0.in 41
7.1.2 PL0_code1.in 41
7.1.3 PL0_code2.in 41
7.1.4 PL0_code3.in 42
7.1.5 PL0_code4.in 42
7.1.6 PL0_code5.in 43
7.2 词法分析结果 44
7.2.1 PL0_code0.in 词法分析结果 44
7.2.2 PL0_code1.in 词法分析结果 45
7.2.3 PL0_code2.in 词法分析结果 45
7.2.4 PL0_code3.in 词法分析结果 45
7.2.5 PL0_code4.in 词法分析结果 46
7.2.6 PL0_code5.in 词法分析结果 46
7.3 语法分析结果 47
7.3.1 PL0_code0.in 语法分析结果 47
7.3.2 PL0_code1.in 语法分析结果 48
7.3.3 PL0_code2.in 语法分析结果 48
7.3.4 PL0_code3.in 语法分析结果 48
7.3.5 PL0_code4.in 语法分析结果 48
7.3.6 PL0_code5.in 语法分析结果 49
7.4 语法树生成结果 49
7.4.1 PL0_code0.in 语法树生成结果 49
7.4.2 PL0_code1.in 语法树生成结果 52
7.4.3 PL0_code2.in 语法树生成结果 53
7.4.4 PL0_code3.in 语法树生成结果 54
7.4.5 PL0_code4.in 语法树生成结果 57
7.4.6 PL0_code5.in 语法树生成结果 65
7.5 中间代码生成结果 70
7.5.1 PL0_code0.in 中间代码生成结果 70
7.5.2 PL0_code1.in 中间代码生成结果 71
7.5.3 PL0_code2.in 中间代码生成结果 72
7.5.4 PL0_code3.in 中间代码生成结果 72
7.5.5 PL0_code4.in 中间代码生成结果 73
7.5.6 PL0_code5.in 中间代码生成结果 76

7.6 解释执行结果 77
7.6.1 PL0_code0.in 解释执行结果 77
7.6.2 PL0_code1.in 解释执行结果 78
7.6.3 PL0_code2.in 解释执行结果 78
7.6.4 PL0_code3.in 解释执行结果 78
7.6.5 PL0_code4.in 解释执行结果 79
7.6.6 PL0_code5.in 解释执行结果 79

1 前言

1.1 设计目的

（1）运用编译原理所学知识，理论联系实际，设计 PL0 语言编译器；
（2）巩固课堂知识；
（3）深化学习内容；

1.2 设计环境

开发环境：
（1）CPU：AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10GHz
（2）内存：16.0GB
（3）操作系统：Windows 10 专业版
（4）集成开发环境：CLion 2020.1 x64
（5）编译工具：CMake 3.15.3
（6）编译器：MinGW 4.3.5
（6）C++标准：C++14

1.3 设计原则

一个优秀的工程不仅要拥有前期设计方便、稳定的特点，还要具有良好的可读性和维护性，基于此，在本工程中提出要遵守的一下设计原则：
（1）项目组织原则：一个优秀的大型项目不能写在一个 cpp 文件中，理应有良好的项目组织架构、目录管理架构，必须做到资源文件和代码文件分离，文档和编码分离。
（2）顶层设计原则：由外而内的设计原则，基于一个总的整体结构，设计子模块，子模块要多采用封装形式。
（3）面向对象设计原则：采用面向对象设计的工程，具有易维护、质量高、效率高、易扩展等优点，可设计出高内聚低耦合的系统结构。

1.4 PL/0 语言文法的 BNF 表示

代码 1-1 PL0 语言文法的 BNF 表示

```
1.  〈程序〉→〈分程序〉.
2.  〈分程序〉→ [〈常量说明部分〉][〈变量说明部分〉][〈过程说明部分〉]〈语句〉
```

```
〈常量说明部分〉 → CONST〈常量定义〉{ ,〈常量定义〉};
〈常量定义〉 → 〈标识符〉=〈无符号整数〉
〈无符号整数〉 → 〈数字〉{〈数字〉}
〈变量说明部分〉 → VAR〈标识符〉{ ,〈标识符〉};
〈标识符〉 → 〈字母〉{〈字母〉|〈数字〉}
〈过程说明部分〉 → 〈过程首部〉〈分程序〉;{〈过程说明部分〉}
〈过程首部〉 → procedure〈标识符〉;
〈语句〉 → 〈赋值语句〉|〈条件语句〉|〈当型循环语句〉|〈过程调用语句〉|〈读语句〉|〈写语句〉|〈复合语句〉|〈空〉
〈赋值语句〉 → 〈标识符〉:=〈表达式〉
〈复合语句〉 → begin〈语句〉{ ; 〈语句〉}〈end〉
〈条件〉 → 〈表达式〉〈关系运算符〉〈表达式〉|odd〈表达式〉
〈表达式〉 → [+|-]〈项〉{〈加减运算符〉〈项〉}
〈项〉 → 〈因子〉{〈乘除运算符〉〈因子〉}
〈因子〉 → 〈标识符〉|〈无符号整数〉|(〈表达式〉)
〈加减运符〉 → +|-
〈乘除运算符〉 → *|/
〈关系运算符〉 → =|#|<|<=|>|>=
〈条件语句〉 → if〈条件〉then〈语句〉
〈过程调用语句〉 → call〈标识符〉
〈当型循环语句〉 → while〈条件〉do〈语句〉
〈读语句〉 → read(〈标识符〉{ , 〈标识符〉})
〈写语句〉 → write(〈表达式〉{, 〈表达式〉})
〈字母〉 → a|b|c…x|y|z
〈数字〉 → 0|1|2…7|8|9
```

1.5 PL/0 编译系统结构

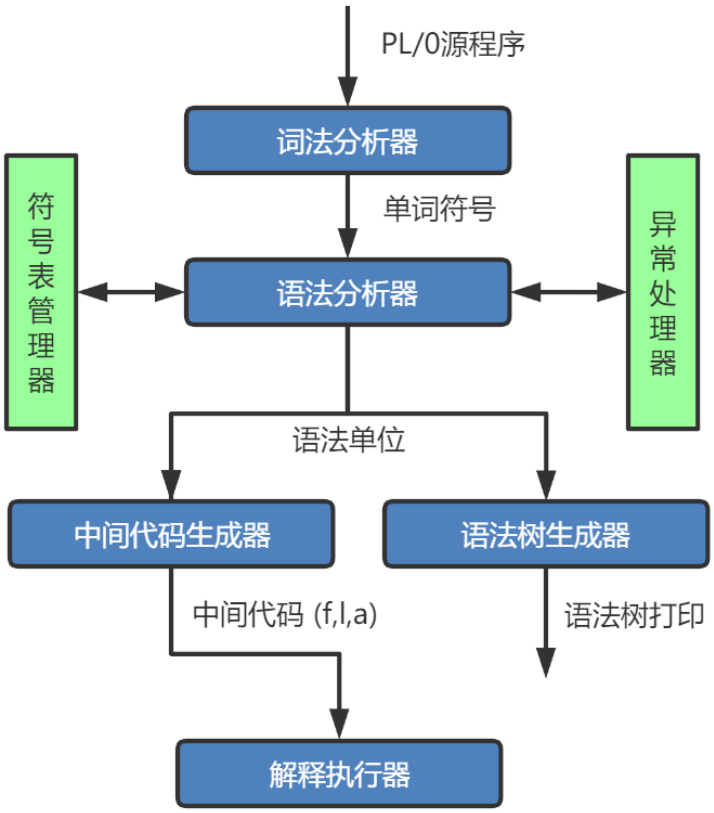

图 1-1 PL0 编译系统结构

2 词法分析器

2.1 任务要求

（1）把关键字、算符、界符称为语言固有的单词，标识符、常量称为用户自定义的单词。为此设置三个全程量：SYM, ID, NUM

- SYM：存放每个单词的类别，为内部编码的表示形式。
- ID：存放用户所定义的标识符的值，即标识符字符串的机内表示。
- NUM：存放用户定义的数。

（2）GETSYM 要完成的任务：

- 滤掉单词间的空格。
- 识别关键字，用查关键字表的方法识别。当单词是关键字时，将对应的类别放在 SYM 中。如 IF 的类别为 IFSYM，THEN 的类别为 THENSYM。
- 识别标识符，标识符的类别为 IDENT，IDENT 放在 SYM 中，标识符本身的值放在 ID 中。关键字或标识符的最大长度是 10。
- 拼数，将数的类别 NUMBER 放在SYM中，数本身的值放在 NUM 中。
- 拼由两个字符组成的运算符，如：>=、<= 等等，识别后将类别存放在 SYM 中。
- 打印源程序，边读入字符边打印。(由于一个单词是由一个或多个字符组成的，所以在词法分析程序 GETSYM 中定义一个读字符过程 GETCH)

2.2 符号类型类的设计

任务要求中定义关键字、算符、界符是语言固有单词，标识符、常量是用户自定义单词。需要设计一个枚举类承载它们，但是 C++的枚举类没法像 Java 一样设计多值组和方法。所以设计一个类 SymbolType 用于承载常量，随后设计一个类 Symbol 来承载具体的 Symbol 及其值。

随后将具体的“BEGIN”、“END”等作为 SymbolType 的类对象实例，使用“const”标记为常量，并且用 namespace“SYMBOL”以防命名空间污染。

代码 2-1 symbolTyle.h

```
/*
 * Copyright 2020 ZhangT. All Rights Reserved.
 * Author: ZhangT
 * Author-Github: github.com/zhangt2333
 * symbolType.h 2020/4/13
 * Comments:
 */
#ifndef PL0_COMPILER_SYMBOLTYPE_H
#define PL0_COMPILER_SYMBOLTYPE_H

```

```cpp
#include <string>
#include <utility>
#include <ostream>

/*
 * 符号类型的承载类，完善枚举类不能实现的多值组
 */
class SymbolType
{
public:
    std::string name;
    int id;
    SymbolType() {}
    SymbolType(std::string name, int id) : name(std::move(name)), id(id) {}

    friend std::ostream &operator<<(std::ostream &os, const SymbolType &type)
    {
        os << "[name: " << type.name << ", id: " << type.id << "]";
        return os;
    }

    bool operator==(const SymbolType &rhs) const
    {
        return name == rhs.name && id == rhs.id;
    }

    bool operator!=(const SymbolType &rhs) const
    {
        return !(rhs == *this);
    }
};

namespace SYMBOL {

const SymbolType/* 关键字 */
                BEGIN       = SymbolType("begin", 1),
                END         = SymbolType("end", 2),
                IF          = SymbolType("if", 3),
                THEN        = SymbolType("then", 4),
                ELSE        = SymbolType("else", 5),
                CONST       = SymbolType("const", 6),
                PROCEDURE   = SymbolType("procedure", 7),
                VAR         = SymbolType("var", 8),
                DO          = SymbolType("do", 9),
                WHILE       = SymbolType("while", 10),
                CALL        = SymbolType("call", 11),
                READ        = SymbolType("read", 12),
                WRITE       = SymbolType("write", 13),
                ODD         = SymbolType("odd", 14),
                /* 算符 */
                EQU         = SymbolType("=", 15),
                LS          = SymbolType("<", 16),
                LEQ         = SymbolType("<=", 17),
                GT          = SymbolType(">", 18),
                GEQ         = SymbolType(">=", 19),
                NEQ         = SymbolType("#", 20),
                ADD         = SymbolType("+", 21),
                SUB         = SymbolType("-", 22),
                MUL         = SymbolType("*", 23),
                DIV         = SymbolType("/", 24),
                /* 界符 */
                CEQU        = SymbolType(":=", 26),
                COMMA       = SymbolType(",", 27),
                SEMIC       = SymbolType(";", 28),
                POINT       = SymbolType(".", 29),
                LBR         = SymbolType("(", 30),
```

```
77.                 RBR         = SymbolType(")", 31),
78.                 /* 标识符, 常量 */
79.                 IDENTIFIER  = SymbolType("identifier", 32),
80.                 NUMBER      = SymbolType("number", 33),
81.                 /* 终结符 */
82.                 END_OF_FILE = SymbolType("$", 34),
83.                 /* 非法符 */
84.                 ILLEGAL     = SymbolType("illegal", 35);
85. }
86. #endif //PL0_COMPILER_SYMBOLTYPE_H
```

2.3 符号类的设计

设计一个类 Symbol 来承载具体的 SymbolType 以及符号值。使用 getter 和 setter 方法封装属性接口。并且使用 unordered_map 设计 KEYWORD_MAP，用于接下来词法分析器的关键字转具体 Symbol 的工具，而避免了使用大量的 if-else 代码做词法分析。

代码 2-2 symbol.h

```
1.  /*
2.   * Copyright 2020 ZhangT. All Rights Reserved.
3.   * Author: ZhangT
4.   * Author-Github: github.com/zhangt2333
5.   * Symbol.h 2020/4/14
6.   * Comments:
7.   */
8.  #ifndef PL0_COMPILER_SYMBOL_H
9.  #define PL0_COMPILER_SYMBOL_H
10.
11.
12. #include <utility>
13. #include <unordered_map>
14. #include <ostream>
15.
16. #include "symbolType.h"
17.
18. /*
19.  * 符号类
20.  */
21. class Symbol
22. {
23. public:
24.     explicit Symbol(SymbolType symbolType): symbolType(std::move(symbolType))
25.     {
26.     }
27.
28.     Symbol(SymbolType symbolType, std::string value): symbolType(std::move(symbolType)), va
    lue(std::move(value))
29.     {
30.     }
31.
32.     Symbol(SymbolType symbolType, int number): symbolType(std::move(symbolType)), number(nu
    mber)
33.     {
34.     }
35.
36.     Symbol(SymbolType symbolType, std::string name, int _number, int _level, int _address):

37.         symbolType(std::move(symbolType)),
38.         value(std::move(name)),
39.         number(_number),
40.         level(_level),
```

```
41.         address(_address)
42.     {
43.     }
44.
45.     friend std::ostream &operator<<(std::ostream &os, const Symbol &symbol)
46.     {
47.         os << symbol.symbolType.name;
48.         if (!symbol.value.empty())
49.             os << '(' << symbol.value << ')';
50.         else if(-1 != symbol.number)
51.             os << '(' << symbol.number << ')';
52.         return os;
53.     }
54.
55.     const SymbolType &getSymbolType() const
56.     {
57.         return symbolType;
58.     }
59.
60.     const std::string &getValue() const
61.     {
62.         return value;
63.     }
64.
65.     int getNumber() const
66.     {
67.         return number;
68.     }
69.
70.     int getLevel() const
71.     {
72.         return level;
73.     }
74.
75.     int getAddress() const
76.     {
77.         return address;
78.     }
79.
80.     void setLevel(int _level)
81.     {
82.         this->level = _level;
83.     }
84.
85.     void setAddress(int _address)
86.     {
87.         this->address = _address;
88.     }
89.
90.     void setValue(const std::string &_value)
91.     {
92.         this->value = _value;
93.     }
94.
95.     void setSymbolType(const SymbolType &_symbolType)
96.     {
97.         this->symbolType = _symbolType;
98.     }
99.
100.    void setNumber(int _number)
101.    {
102.        this->number = _number;
103.    }
104. private:
105.    SymbolType symbolType;
106.    std::string value = "";
```

```
107.     int number = -1;
108.     int level = -1;
109.     int address = -1;
110. };
111.
112. namespace SYMBOL {
113.
114. const std::unordered_map<std::string, Symbol> KEYWORD_MAP({
115.     /* 关键词 */
116.     {SYMBOL::BEGIN.name,     Symbol(SYMBOL::BEGIN)},
117.     {SYMBOL::END.name,       Symbol(SYMBOL::END)},
118.     {SYMBOL::IF.name,        Symbol(SYMBOL::IF)},
119.     {SYMBOL::THEN.name,      Symbol(SYMBOL::THEN)},
120.     {SYMBOL::ELSE.name,      Symbol(SYMBOL::ELSE)},
121.     {SYMBOL::CONST.name,     Symbol(SYMBOL::CONST)},
122.     {SYMBOL::PROCEDURE.name, Symbol(SYMBOL::PROCEDURE)},
123.     {SYMBOL::VAR.name,       Symbol(SYMBOL::VAR)},
124.     {SYMBOL::DO.name,        Symbol(SYMBOL::DO)},
125.     {SYMBOL::WHILE.name,     Symbol(SYMBOL::WHILE)},
126.     {SYMBOL::CALL.name,      Symbol(SYMBOL::CALL)},
127.     {SYMBOL::READ.name,      Symbol(SYMBOL::READ)},
128.     {SYMBOL::WRITE.name,     Symbol(SYMBOL::WRITE)},
129.     {SYMBOL::ODD.name,       Symbol(SYMBOL::ODD)}
130. });
131.
132. const std::unordered_map<std::string, Symbol> SIGN_MAP({
133.      /* 算符 */
134.     {SYMBOL::EQU.name,   Symbol(SYMBOL::EQU)},
135.     {SYMBOL::LS.name,    Symbol(SYMBOL::LS)},
136.     {SYMBOL::LEQ.name,   Symbol(SYMBOL::LEQ)},
137.     {SYMBOL::GT.name,    Symbol(SYMBOL::GT)},
138.     {SYMBOL::GEQ.name,   Symbol(SYMBOL::GEQ)},
139.     {SYMBOL::NEQ.name,   Symbol(SYMBOL::NEQ)},
140.     {SYMBOL::ADD.name,   Symbol(SYMBOL::ADD)},
141.     {SYMBOL::SUB.name,   Symbol(SYMBOL::SUB)},
142.     {SYMBOL::MUL.name,   Symbol(SYMBOL::MUL)},
143.     {SYMBOL::DIV.name,   Symbol(SYMBOL::DIV)},
144.     /* 界符 */
145.     {SYMBOL::CEQU.name,  Symbol(SYMBOL::CEQU)},
146.     {SYMBOL::COMMA.name, Symbol(SYMBOL::COMMA)},
147.     {SYMBOL::SEMIC.name, Symbol(SYMBOL::SEMIC)},
148.     {SYMBOL::POINT.name, Symbol(SYMBOL::POINT)},
149.     {SYMBOL::LBR.name,   Symbol(SYMBOL::LBR)},
150.     {SYMBOL::RBR.name,   Symbol(SYMBOL::RBR)}
151. });
152.
153.
154. }
155.
156.
157. #endif //PL0_COMPILER_SYMBOL_H
```

2.4 词法分析器类的设计

按照面向对象思想完成词法分析任务，需要设计一个词法分析器 Lexer，传入一个输入流的父类 istream 的对象，对于外部调用者来说，传入文件流、字符串流都是便于使用的，很好地利用了面向对象中多态的性质。随后 Lexer 向外提供接口“getSymbol”即可，返回值是符号的包装类“Symbol”的实例对象。描述的类声明代码如下：

代码 2-3 lexer.h

```
/*
 * Copyright 2020 ZhangT. All Rights Reserved.
 * Author: ZhangT
 * Author-Github: github.com/zhangt2333
 * lexer.h 2020/4/13
 * Comments:
 */
#ifndef PL0_COMPILER_LEXER_H
#define PL0_COMPILER_LEXER_H

#include <istream>
#include <vector>
#include "symbol.h"

/**
 * 词法分析器
 */
class Lexer
{
public:
    Lexer(std::istream &_inputStream) : inputStream(_inputStream)
    {}
    Symbol getSymbol();
    bool isEOF();

private:
    std::istream &inputStream;
    char get();
    char peek();
};


#endif //PL0_COMPILER_LEXER_H
```

其中，私有方法“get”和“peek”用于包装从 istream 中读取字符的操作。具体的实现代码如下：

代码 2-4　lexer.cpp

```
/*
 * Copyright 2020 ZhangT. All Rights Reserved.
 * Author: ZhangT
 * Author-Github: github.com/zhangt2333
 * lexer.cpp 2020/4/13
 * Comments:
 */

//#define DEBUG    // 调试模式

#include <iostream>
#include <cctype>
#include "utils.h"
#include "lexer.h"

char Lexer::get()
{
    char ch = inputStream.get();
    logs("LOG[Lexer: get]: ", ch);
    return ch;
}

char Lexer::peek()
{
    char ch = inputStream.peek();
```

```
26.     logs("LOG[Lexer: peek]: ", ch);
27.     return ch;
28. }
29.
30. bool Lexer::isEOF()
31. {
32.     return inputStream.eof();
33. }
34.
35. Symbol Lexer::getSymbol()
36. {
37.     if (isEOF())
38.         return Symbol(SYMBOL::END_OF_FILE);
39.
40.     while(isspace(peek()))
41.         get();
42.
43.     std::vector<char> strToken;
44.     if (isalpha(peek()))
45.     {
46.         do
47.         {
48.             strToken.emplace_back(get());
49.         }while(isalpha(peek()) || isdigit(peek()));
50.         std::string value(strToken.begin(), strToken.end());
51.         auto it = SYMBOL::KEYWORD_MAP.find(value);
52.         return it != SYMBOL::KEYWORD_MAP.end() ? it->second : Symbol(SYMBOL::IDENTIFIER, va
    lue);
53.     } else if (isdigit(peek()))
54.     {
55.         do
56.         {
57.             strToken.emplace_back(get());
58.         }while(isdigit(peek()));
59.         int number = vectorToInt(strToken);
60.         return Symbol(SYMBOL::NUMBER, number);
61.     }
62.     std::string str(1, get());
63.     char pk = peek();
64.     switch (str[0])
65.     {
66.         case '<': case '>': case ':':
67.             if(pk == '=') str += get();
68.     }
69.     auto it = SYMBOL::SIGN_MAP.find(str);
70.     logs("LOG[getSymbol: ToFind]", str, "\n");
71.     return it != SYMBOL::SIGN_MAP.end() ? it->second : Symbol(SYMBOL::ILLEGAL, str);
72. }
```

代码中黄色高亮的部分，从上到下分别任务要求中实现的点，即滤掉词间空格、识别关键字或者用户标识符、拼数、拼两字符组成的运算符。

2.5 测试代码

分别用“../test/PL0_code0.in”、…、“../test/PL0_code6.in”作为命令行启动参数，调用下面的 Lexer 测试代码，可以测试词法分析器是否能正常工作。具体的输出结果，放置于附录中。

代码 2-5 Lexer 测试代码

```cpp
#include <iostream>
#include <fstream>
#include "lexer.h"

/* 词法分析测试代码 */
int main(int argc, char **argv)
{
    std::ifstream fin(argv[1]);
    Lexer lexer(fin);
    while (!lexer.isEOF())
    {
        Symbol symbol = lexer.getSymbol();
        std::cout << symbol << ' ';
        if (SYMBOL::SEMIC == symbol.getSymbolType())
            std::cout << std::endl;
    }
    return 0;
}
```

3 语法分析器

3.1 任务要求

PL/0 编译程序采用一遍扫描的方法，所以语法分析和代码生成都在 BLOCK 中完成。BLOCK 的工作分为两步，分为“说明部分的处理”和“语句处理和代码生成”，本章重点解决“说明部分的处理”，即语法分析部分。

说明部分的处理任务就是对每个过程（包括主程序，可以看成是一个主过程）的说明对象造名字表。填写所在层次（主程序是 0 层，在主程序中定义的过程是 1 层，随着嵌套的深度增加而层次数增大。PL/0 最多允许 3 层），标识符的属性和分配的相对地址等。标识符的属性不同则填写的信息不同。

所造的表放在全程量一维数组 TABLE 中，TX 为指针，数组元素为结构体类型数据。LEV 给出层次，DX 给出每层的局部量的相对地址，每说明完一个变量后 DX 加 1。

例如如下一个过程的说明部分，生成的符号表为：

代码 3-1 过程说明部分的例子

```
1.  const a=35,b=49;
2.  var c,d,e;
3.  procedure p;
4.  var g;
```

表 3-1 过程说明部分的 TABLE 表

TX0 ->			
NAME: a NAME: b NAME: c NAME: d NAME: e NAME: p	KIND: CONSTANT KIND: CONSTANT KIND: VARIABLE KIND: VARIABLE KIND: VAEIABLE KIND: PROCEDURE	VAL: 35 VAL: 49 LEVEL: LEV LEVEL: LEV LEVEL: LEV LEVEL: LEV	 ADR: DX ADR: DX+1 ADR: DX+2 ADR:
TX1 ->			
NAME: g …	KIND: VARIABLE …	LEVEL: LEV+1 …	ADR: DX …

对于过程名的 ADR 域，是在过程体的目标代码生成后返填过程体的入口地址。

TABLE 表的索引 TX 和层次单元 LEV 都是以 BLOCK 的参数形式出现，在主程序调用 BLOCK 时实参的值为 0。每个过程的相对起始位置在 BLOCK 内置初值 DX=3。

3.2 异常处理器的设计

一个优秀的工程，理应有良好的异常反馈/处理机制，在本项目中，主要反应在语法分析出错时，能便于定位和解决错误。基于此，设计了异常处理器，设计期间迭代了两个版本，

一个是与 if-else 搭配的异常处理器设计，属于提前判断式的异常处理，最终的版本是采用类似“assert”的思路和精髓，使得异常处理与逻辑代码便于分开，不会降低逻辑代码的可读性。此外，设计异常的枚举类便于异常的拓展和使用。

代码 3-2　exceptionHandler.h

```
/*
 * Copyright 2020 ZhangT. All Rights Reserved.
 * Author: ZhangT
 * Author-Github: github.com/zhangt2333
 * exceptionHandler.h 2020/4/21
 * Comments:
 */
#ifndef PL0_COMPILER_EXCEPTIONHANDLER_H
#define PL0_COMPILER_EXCEPTIONHANDLER_H

#include <iostream>

namespace EXCEPTION {
    enum ExceptionEnum{
        EXTRA_CHARACTERS,
        MISSING_SEMICOLON,
        MISSING_IDENTIFIER,
        DUPLICATE_IDENTIFIER,
        MISSING_EQUAL,
        MISSING_CEQUAL,
        MISSING_NUMBER,
        MISSING_THEN,
        MISSING_DO,
        MISSING_LBR,
        MISSING_RBR,
        MISSING_END,
        MISSING_SEMIC,
        NOT_A_VAR,
        NOT_A_PROCUDURE,
        NEVER_DECLARE,
    };
    std::unordered_map<ExceptionEnum, std::string> EXCEPTION_ENUM_MAP({
        {EXTRA_CHARACTERS, "There is extra char after '.' !"},
        {EXTRA_CHARACTERS, "Here is missing a semicolon !"},
        {MISSING_SEMICOLON, "Here is missing a identifier !"},
        {MISSING_IDENTIFIER, "MISSING_IDENTIFIER !"},
        {DUPLICATE_IDENTIFIER, "DUPLICATE_IDENTIFIER"},
        {MISSING_EQUAL, "MISSING_EQUAL"},
        {MISSING_CEQUAL, "MISSING_CEQUAL"},
        {MISSING_NUMBER, "MISSING_NUMBER"},
        {MISSING_THEN, "MISSING_THEN"},
        {MISSING_DO, "MISSING_DO"},
        {MISSING_LBR, "MISSING_LBR"},
        {MISSING_RBR, "MISSING_RBR"},
        {MISSING_END, "MISSING_END"},
        {MISSING_SEMIC, "MISSING_SEMIC"},
        {NOT_A_VAR, "NOT_A_VAR"},
        {NOT_A_PROCUDURE, "NOT_A_PROCUDURE"},
        {NEVER_DECLARE, "NEVER_DECLARE"}
    });

};
void ASSERT(bool _, EXCEPTION::ExceptionEnum parserExceptionEnum)
{ // TODO: 更详细的异常信息、更方便的异常处理
    if (_) return;
    std::cout << "--------------------" << EXCEPTION::EXCEPTION_ENUM_MAP[parserExceptionEnum
] << std::endl;
```

```
57.     exit(1);
58. }
59.
60.
61. #endif //PL0_COMPILER_EXCEPTIONHANDLER_H
```

列举使用异常处理器的语法分析器代码如下：

代码 3-3　parser.cpp 部分代码

```
1.  // 〈常量定义〉 → 〈标识符〉=〈无符号整数〉
2.  // 〈无符号整数〉 → 〈数字〉{〈数字〉}
3.  void Parser::constDefine()
4.  {
5.      logs("LOG[Parser: constDefine]");
6.      syntaxTree.begin("[constDefine]");
7.
8.      ASSERT(SYMBOL::IDENTIFIER == nowSymbolType, EXCEPTION::MISSING_IDENTIFIER); // 异常处
    理：缺少标识符
9.      std::string constName = nowSymbol.getValue();
10.     ASSERT(!symbolTable.inTable(constName), EXCEPTION::DUPLICATE_IDENTIFIER); // 特判标识符重
    复声明
11.     advance();                                  // 跳过标识符
12.     ASSERT(SYMBOL::EQU == nowSymbolType, EXCEPTION::MISSING_EQUAL); // 异常处理：缺少 '='
13.     advance();                                  // 跳过 '='
14.     ASSERT(SYMBOL::NUMBER == nowSymbolType, EXCEPTION::MISSING_NUMBER); // 异常处理：缺少无符
    号整数
15.     symbolTable.addSymbol(SYMBOL::CONST, constName, nowSymbol.getNumber(), level, address);

16.     advance();                                  // 跳过 无符号整数
17.
18.     syntaxTree.end();
19. }
```

3.3　符号表及其管理器的设计

符号表，或者说 TABLE 表，要解决的问题是能分层管理 Symbol，这里的管理指的是添加新 Symbol、查找 Symbol、判断 Symbol 是否已存在、查找一个 Procedure 的 Symbol。

所以声明文件中有两个类，一个是 SymbolTable，主要是用 unordered_map 存储符号名到 Symbol 的映射，然后是 list 存储所有该表的 Symbol，向外提供“inTable”和“addSymbol”接口。

Procedure 有层次递进关系，在 SymbolTableManager 中使用栈这个数据结构去存储 SymbolTable，同时使用数组存储所有的 SymbolTable。具体的声明文件如下：

代码 3-4　symbolTable.h

```
1.  /*
2.   * Copyright 2020 ZhangT. All Rights Reserved.
3.   * Author: ZhangT
4.   * Author-Github: github.com/zhangt2333
5.   * symbolTable.h 2020/4/21
6.   * Comments:
7.   */
8.  #ifndef PL0_COMPILER_SYMBOLTABLE_H
9.  #define PL0_COMPILER_SYMBOLTABLE_H
10.
11. #include <unordered_map>
12. #include <list>
```

```
13. #include "symbol.h"
14.
15. class SymbolTable
16. {
17. public:
18.     std::unordered_map<std::string, Symbol*> mp;
19.     std::list<Symbol*> lst;
20.     bool inTable(const std::string& name);
21.     void addSymbol(Symbol* symbol);
22. };
23.
24. class SymbolTableManager
25. {
26.     std::list<SymbolTable*> symbolTableStack;  // 符号表 栈
27.     std::list<SymbolTable*> symbolTableList;   // 符号表 数组
28. public:
29.     SymbolTableManager()
30.     {
31.         pushTable();
32.     }
33.
34.     void addSymbol(const SymbolType& symbolType, const std::string &name, int number, int le
    vel, int address);
35.     Symbol *getLastProcedure();
36.     bool inTable(const std::string& name);
37.     Symbol *getSymbol(const std::string &name);
38.     void pushTable();
39.     void popTable();
40.     void printTables();
41. };
42.
43.
44. #endif // PL0_COMPILER_SYMBOLTABLE_H
```

具体的实现代码，比如添加 Symbol，建立新表、弹出栈顶的表，都是对 STL 的操作。需要注意的是查找上层 Procedure 的时候需要倒序找，因为程序的调用是逐层递进的，那么找上层调用者即倒序找。

打印符号表的程序使用了“std::setiosflags(std::ios::left)”做对齐和“std::setfill(' ')”做填充，使得打印出的符号表能对齐边界，整齐好看。

代码 3-5　symbolTable.cpp

```
1.  /*
2.   * Copyright 2020 ZhangT. All Rights Reserved.
3.   * Author: ZhangT
4.   * Author-Github: github.com/zhangt2333
5.   * symbolTable.cpp 2020/4/21
6.   * Comments:
7.   */
8.
9.
10. #include <iostream>
11. #include <iomanip>
12. #include "symbolTable.h"
13.
14. bool SymbolTable::inTable(const std::string &name)
15. {
16.     return mp.find(name) != mp.end();
17. }
18.
19. void SymbolTable::addSymbol(Symbol *symbol)
20. {
```

```
21.     mp[symbol->getValue()] = symbol;
22.     lst.push_back(symbol);
23. }
24. 
25. void SymbolTableManager::addSymbol(const SymbolType &symbolType, const std::string &name, i
    nt number, int level, int address)
26. {
27.     symbolTableStack.back()->addSymbol(new Symbol(symbolType, name, number, level, address)
    );
28. }
29. 
30. Symbol *SymbolTableManager::getLastProcedure()
31. {
32.     for (auto it = symbolTableList.rbegin(); it != symbolTableList.rend(); it++)
33.         for (auto it2 =  (*it)->lst.rbegin(); it2 != (*it)->lst.rend(); it2++)
34.             if (SYMBOL::PROCEDURE == (*it2)->getSymbolType())
35.                 return *it2;
36.     return nullptr;
37. }
38. 
39. bool SymbolTableManager::inTable(const std::string &name)
40. {
41.     return symbolTableStack.back()->inTable(name);
42. }
43. 
44. Symbol *SymbolTableManager::getSymbol(const std::string &name)
45. {
46.     for (auto *t : symbolTableStack)
47.     {
48.         auto it = t->mp.find(name);
49.         if (it != t->mp.end())
50.             return it->second;
51.     }
52.     return nullptr;
53. }
54. 
55. void SymbolTableManager::pushTable()
56. {
57.     symbolTableStack.push_back(new SymbolTable());
58.     symbolTableList.push_back(symbolTableStack.back());
59. }
60. 
61. void SymbolTableManager::popTable()
62. {
63.     symbolTableStack.pop_back();
64. }
65. 
66. void SymbolTableManager::printTables()
67. {
68.     int idx = 0;
69.     for (auto *t : symbolTableList)
70.     {
71.         std::cout << "TX" << idx++ << "->" << std::endl;
72.         for (auto *symbol : t->lst)
73.         {
74.             std::cout << "NAME: ";
75.             std::cout << std::setiosflags(std::ios::left) << std::setfill(' ') << std::setw
    (10) << symbol->getValue();
76.             std::cout << "KIND: ";
77.             std::cout << std::setiosflags(std::ios::left) << std::setfill(' ') << std::setw
    (10)
78.                       << symbol->getSymbolType().name;
79.             if (SYMBOL::CONST == symbol->getSymbolType())
80.             {
81.                 std::cout << "VAL: ";
```

```
                std::cout << std::setiosflags(std::ios::left) << std::setfill(' ') << std::
    setw(10)
                          << symbol->getNumber();
            } else
            {
                std::cout << "LEVEL: ";
                std::cout << std::setiosflags(std::ios::left) << std::setfill(' ') << std::
    setw(10)
                          << symbol->getLevel();
                std::cout << "ADR: ";
                std::cout << std::setiosflags(std::ios::left) << std::setfill(' ') << std::
    setw(10)
                          << symbol->getAddress();
            }
            std::cout << std::endl;
        }
    }
}
```

3.4 语法分析器的设计

对于每个产生式的左部，设计一个分析函数，如“program()”、“ifStatement()”，体现“问题分解”的计算思维。例如以下描述图中将程序分为一个分程序进行语法分析，将分程序分为常量说明部分、变量说明部分、过程说明部分、语句进行语法分析。

图 3-1 程序语法描述图

图 3-2 分程序语法描述图

具体到常量说明部分里面又有细分常量定义部分，语法分析器代码的说明部分如下：

代码 3-6 parser.h

```
/*
 * Copyright 2020 ZhangT. All Rights Reserved.
 * Author: ZhangT
 * Author-Github: github.com/zhangt2333
 * parser.h 2020/4/19
 * Comments:
 */
#ifndef PL0_COMPILER_PARSER_H
#define PL0_COMPILER_PARSER_H


#include "lexer.h"
#include "symbolTable.h"
#include "code.h"
#include "syntaxTree.h"

class Parser
{
    Lexer &lexer;
    int level = 0;
    int address = 3;
```

```
22.     SymbolType nowSymbolType = SYMBOL::ILLEGAL;
23.     Symbol nowSymbol = Symbol(nowSymbolType);
24. public:
25.     SymbolTableManager symbolTable;
26.     std::vector<Code*> codeTable;
27.     SyntaxTree syntaxTree;
28.     explicit Parser(Lexer &lexer);
29.     void printTables() { symbolTable.printTables(); }
30.     void printCode();
31.     // 〈程序〉→〈分程序〉.
32.     void program();
33. 
34. private:
35.     void advance();
36.     void addressReset();
37.     int getCodeAddress();
38. 
39.     // 〈分程序〉→ [〈常量说明部分〉][〈变量说明部分〉][〈过程说明部分〉]〈语句〉
40.     void subProbgram();
41.     // 〈常量说明部分〉 → const〈常量定义〉{ ,〈常量定义〉};
42.     void constDeclare();
43.     // 〈常量定义〉 → 〈标识符〉=〈无符号整数〉
44.     void constDefine();
45.     // 〈变量说明部分〉 → var〈标识符〉{ ,〈标识符〉};
46.     void varDeclare();
47.     // 〈过程说明部分〉 → 〈过程首部〉〈分程序〉; {〈过程说明部分〉}
48.     void procDeclare();
49.     // 〈语句〉 → 〈赋值语句〉|〈条件语句〉|〈当型循环语句〉|〈过程调用语句〉|〈读语句〉|〈写语句〉|〈复合语句〉|〈空〉
50.     void statement();
51.     // 〈赋值语句〉 → 〈标识符〉:=〈表达式〉
52.     bool assignStatement();
53.     // 〈条件语句〉 → if〈条件〉then〈语句〉
54.     bool ifStatement();
55.     // 〈当型循环语句〉 → while〈条件〉do〈语句〉
56.     bool whileStatement();
57.     // 〈过程调用语句〉 → call〈标识符〉
58.     bool callStatement();
59.     // 〈读语句〉 → read(〈标识符〉{ , 〈标识符〉})
60.     bool readStatement();
61.     // 〈写语句〉 → write(〈表达式〉{, 〈表达式〉})
62.     bool writeStatement();
63.     // 〈复合语句〉 → begin〈语句〉{ ; 〈语句〉}〈end〉
64.     bool beginStatement();
65.     // 〈分号〉
66.     void semicolon();
67.     // ) 右括号
68.     void rightBracket();
69.     // 〈表达式〉 → [+|-]〈项〉{〈加减运算符〉〈项〉}
70.     void expression();
71.     // 〈项〉 → 〈因子〉{〈乘除运算符〉〈因子〉}
72.     void term();
73.     // 〈因子〉 → 〈标识符〉|〈无符号整数〉|(〈表达式〉)
74.     void factor();
75.     // 〈条件〉 → 〈表达式〉〈关系运算符〉〈表达式〉|odd〈表达式〉
76.     void condition();
77. 
78. };
79. 
80. #endif //PL0_COMPILER_PARSER_H
```

具体的语法分析实现，即判断当前语法分析进展的情况下，根据递归下降分析法，展望（特判）需要的符号，并且进入下一阶段。如下图中，常量说明部分、变量说明部分、过程

说明部分都已被详细描述，可以根据产生式展望需要匹配的右部符号。

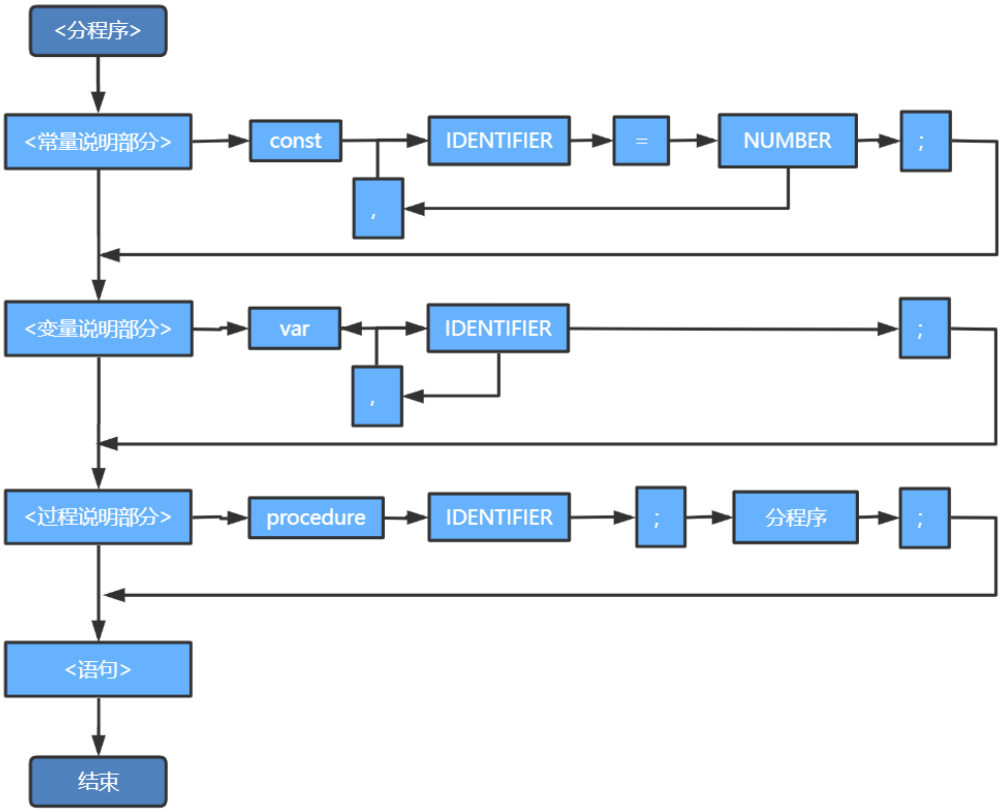

图 3-3 分程序详细语法描述图

如下图，在产生式“语句”的实现中，有多条路径，此时的设计是使用 bool 作为赋值语句、条件语句等产生式分析程序的返回值，有真则停。并且在具体的赋值语句等产生式分析程序中，需要根据展望的语句开头符号特判是否进入该分析程序。

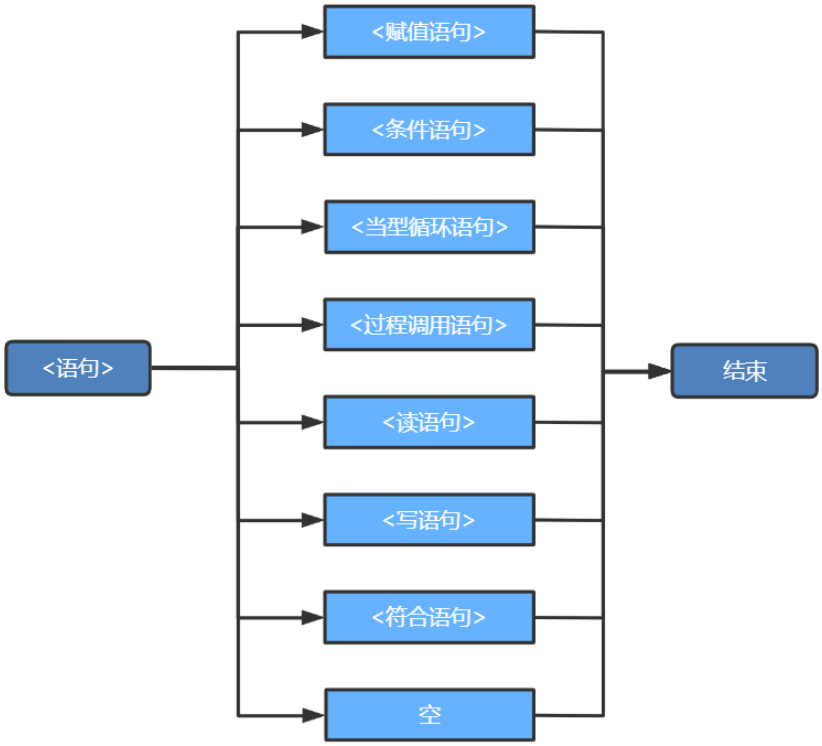

图 3-4 语句的语法描述

语法分析的框架已经如上分析清楚，对于词法分析器 Lexer 的使用，分装了“advance”接口，将下一 Symbol 取到成员变量“nowSymbol”中，并且单独取出“nowSymbolType”，便于语法分析取符号类型。

仅包含语法分析和使用符号表管理的语法分析器 Parser 的具体代码如下：

代码 3-7　parser.cpp 仅含语法分析部分

```
/*
 * Copyright 2020 ZhangT. All Rights Reserved.
 * Author: ZhangT
 * Author-Github: github.com/zhangt2333
 * parser.cpp 2020/4/19
 * Comments:
 */
#define DEBUG    // 调试模式


#include "parser.h"
#include "exceptionHandler.h"
#include "utils.h"


Parser::Parser(Lexer &lexer) : lexer(lexer)
{
    advance();
}

// 取下一个 词
void Parser::advance()
{
    nowSymbol = lexer.getSymbol();
    nowSymbolType = nowSymbol.getSymbolType();
    logs("LOG[Parser: advance] ", nowSymbol);
}

// 每个过程的相对起始位置在 BLOCK 内置初值 DX=3
void Parser::addressReset()
{
    address = 3;
}

int Parser::getCodeAddress()
{
    return codeTable.size();
}

// 〈程序〉→〈分程序〉.
void Parser::program()
{
    logs("LOG[Parser: program]");
    /* 语义分析 */
    subProbgram();                          // 〈分程序〉
    ASSERT(SYMBOL::POINT == nowSymbolType, EXCEPTION::MISSING_SEMICOLON); // 异常处理: 缺少 '.'
    advance();                              // 跳过 '.'
    ASSERT(lexer.isEOF(), EXCEPTION::EXTRA_CHARACTERS); // 异常处理: 文件尾 '.' 后还有字符
}

// 〈分程序〉→ [〈常量说明部分〉][〈变量说明部分〉][〈过程说明部分〉]〈语句〉
void Parser::subProbgram()
{
    logs("LOG[Parser: subProbgram]");
    /* 语义分析 */
```

```
56.     constDeclare(); // [〈常量说明部分〉]
57.     varDeclare();   // [〈变量说明部分〉]
58.     procDeclare();  // [〈过程说明部分〉]
59.     statement();    //  〈语句〉
60. }
61.
62. //  〈常量说明部分〉 → const〈常量定义〉{ ,〈常量定义〉};
63. void Parser::constDeclare()
64. {
65.     logs("LOG[Parser: constDeclare]");
66.     if (SYMBOL::CONST == nowSymbolType)
67.     { // const
68.         advance();     // 跳过 'const'
69.         constDefine();
70.         while (SYMBOL::COMMA == nowSymbolType)
71.         {
72.             advance(); // 跳过 ','
73.             constDefine();
74.         }
75.         semicolon();
76.     }
77. }
78.
79. //  〈常量定义〉 → 〈标识符〉=〈无符号整数〉
80. //  〈无符号整数〉 → 〈数字〉{〈数字〉}
81. void Parser::constDefine()
82. {
83.     logs("LOG[Parser: constDefine]");
84.     ASSERT(SYMBOL::IDENTIFIER == nowSymbolType, EXCEPTION::MISSING_IDENTIFIER); // 异常处
    理: 缺少标识符
85.     std::string constName = nowSymbol.getValue();
86.     ASSERT(!symbolTable.inTable(constName), EXCEPTION::DUPLICATE_IDENTIFIER); // 特判标识符重
    复声明
87.     advance();                                  // 跳过标识符
88.     ASSERT(SYMBOL::EQU == nowSymbolType, EXCEPTION::MISSING_EQUAL); // 异常处理: 缺少 '='
89.     advance();                                  // 跳过 '='
90.     ASSERT(SYMBOL::NUMBER == nowSymbolType, EXCEPTION::MISSING_NUMBER); // 异常处理: 缺少无符
    号整数
91.     symbolTable.addSymbol(SYMBOL::CONST, constName, nowSymbol.getNumber(), level, address);
92.     advance();                                  // 跳过 无符号整数
93. }
94.
95. //  〈变量说明部分〉 → var〈标识符〉{ ,〈标识符〉};
96. void Parser::varDeclare()
97. {
98.     logs("LOG[Parser: varDeclare]");
99.     if (SYMBOL::VAR == nowSymbolType)
100.     {
101.         do
102.         {
103.             advance();                                  // 跳过 'var' or ','
104.             ASSERT(SYMBOL::IDENTIFIER == nowSymbolType, EXCEPTION::MISSING_IDENTIFIER); //
    异常处理: 缺少标识符
105.             std::string varName = nowSymbol.getValue();
106.             ASSERT(!symbolTable.inTable(varName), EXCEPTION::DUPLICATE_IDENTIFIER); // 特判
    标识符重复声明
107.             symbolTable.addSymbol(SYMBOL::VAR, varName, 0, level, address);
108.             address++;                                  // 地址++
109.             advance();                                  // 跳过 标识符
110.         } while (SYMBOL::COMMA == nowSymbolType);
111.         semicolon();                                    // 跳过 ';'
112.     }
113. }
114.
115. //  〈过程说明部分〉 → 〈过程首部〉〈分程序〉;{〈过程说明部分〉}
```

```
116. // 〈过程首部〉 → procedure〈标识符〉;
117. void Parser::procDeclare()
118. {
119.     logs("LOG[Parser: procDeclare]");
120.     while (SYMBOL::PROCEDURE == nowSymbolType)
121.     {
122.         advance();                                  // 跳过 'procedure'
123.
124.         ASSERT(SYMBOL::IDENTIFIER == nowSymbolType, EXCEPTION::MISSING_IDENTIFIER); // 异常处理: 缺少标识符
125.         std::string procName = nowSymbol.getValue();
126.         ASSERT(!symbolTable.inTable(procName), EXCEPTION::DUPLICATE_IDENTIFIER);    // 特判标识符重复声明
127.
128.         symbolTable.addSymbol(SYMBOL::PROCEDURE, procName, 0, level, address);
129.         address++;
130.         advance();                                  // 跳过 标识符
131.         semicolon();                                // 跳过 ';'
132.         /* 进入〈分程序〉*/
133.         level++;
134.         symbolTable.pushTable();
135.         subProbgram(); // 分程序
136.         symbolTable.popTable();
137.         level--;
138.         semicolon();                               // 跳过 ';'
139.     }
140. }
141.
142. // 〈语句〉 → 〈赋值语句〉|〈条件语句〉|〈当型循环语句〉|〈过程调用语句〉|〈读语句〉|〈写语句〉|〈复合语句〉|〈空〉
143. void Parser::statement()
144. {
145.     logs("LOG[Parser: statement]");
146.     if(assignStatement() ||
147.        ifStatement()     ||
148.        whileStatement()  ||
149.        callStatement()   ||
150.        writeStatement()  ||
151.        readStatement()   ||
152.        beginStatement())
153.         return;
154. }
155.
156. // 〈赋值语句〉 → 〈标识符〉:=〈表达式〉
157. bool Parser::assignStatement()
158. {
159.     logs("LOG[Parser: assignStatement]");
160.     if (SYMBOL::IDENTIFIER == nowSymbolType)
161.     {
162.         std::string varName = nowSymbol.getValue();
163.         Symbol *symbol = symbolTable.getSymbol(varName);
164.         ASSERT(nullptr != symbol, EXCEPTION::MISSING_IDENTIFIER);          // 异常处理: 缺少标识符
165.         ASSERT(SYMBOL::VAR == symbol->getSymbolType(), EXCEPTION::NOT_A_VAR); // 异常处理: 并非变量
166.         advance();                                      // 跳过 标识符
167.         ASSERT(SYMBOL::CEQU == nowSymbolType, EXCEPTION::MISSING_CEQUAL); // 异常处理: 缺少赋值号
168.         advance();                                      // 跳过 ':='
169.         expression();
170.         return true;
171.     }
172.     return false;
173. }
174.
175. // 〈条件语句〉 → if〈条件〉then〈语句〉
```

```
176. bool Parser::ifStatement()
177. {
178.     logs("LOG[Parser: ifStatement]");
179.     if (SYMBOL::IF == nowSymbolType)
180.     {
181.         advance();                          // 跳过 'IF'
182.         condition();
183.         ASSERT(SYMBOL::THEN == nowSymbolType, EXCEPTION::MISSING_THEN); // 缺少 THEN
184.         advance();                          // 跳过 'THEN'
185.         statement();
186.         return true;
187.     }
188.     return false;
189. }
190.
191. // 〈当型循环语句〉 → while〈条件〉do〈语句〉
192. bool Parser::whileStatement()
193. {
194.     logs("LOG[Parser: whileStatement]");
195.     if (SYMBOL::WHILE == nowSymbolType)
196.     {
197.         /* 语义分析 */
198.         advance();                           // 跳过 'while'
199.         condition();
200.         ASSERT(SYMBOL::DO == nowSymbolType, EXCEPTION::MISSING_DO); // 缺少 DO
201.         advance(); // 跳过 'do'
202.         Code *jpc = new Code(CODE::JPC);// 条件转移代码，地址回填
203.         codeTable.push_back(jpc);
204.         statement();
205.         return true;
206.     }
207.     return false;
208. }
209.
210. // 〈过程调用语句〉 → call〈标识符〉
211. bool Parser::callStatement()
212. {
213.     logs("LOG[Parser: callStatement]");
214.     if (SYMBOL::CALL == nowSymbolType)
215.     {
216.         /* 语义分析 */
217.         advance();                                  // 跳过 'CALL'
218.         ASSERT(SYMBOL::IDENTIFIER == nowSymbolType, EXCEPTION::MISSING_IDENTIFIER);// 异常处
    理: 缺少标识符
219.         std::string procName = nowSymbol.getValue();
220.         Symbol *symbol = symbolTable.getSymbol(procName);
221.         ASSERT(nullptr != symbol, EXCEPTION::NOT_A_PROCUDURE); // 并非过程
222.         advance();                                  // 跳过 '标识符'
223.         return true;
224.     }
225.     return false;
226. }
227.
228. // 〈读语句〉 → read(〈标识符〉{ , 〈标识符〉})
229. bool Parser::readStatement()
230. {
231.     logs("LOG[Parser: readStatement]");
232.     if (SYMBOL::READ == nowSymbolType)
233.     {
234.         /* 语义分析 */
235.         advance();                                      // 跳过 'read'
236.         ASSERT(SYMBOL::LBR == nowSymbolType, EXCEPTION::MISSING_LBR); // 异常处理: 缺少'('
237.         do
238.         {
239.             advance();                                  // 跳过 '('、','
```

```
240.             ASSERT(SYMBOL::IDENTIFIER == nowSymbolType, EXCEPTION::MISSING_IDENTIFIER); //
    异常处理：缺少标识符
241.             std::string varName = nowSymbol.getValue();
242.             Symbol *symbol = symbolTable.getSymbol(varName);
243.             ASSERT(nullptr != symbol && SYMBOL::VAR == symbol->getSymbolType(), EXCEPTION::
    NOT_A_VAR); // 并非变量
244.             advance(); // 跳过 标识符
245.         } while (SYMBOL::COMMA == nowSymbolType);
246.         rightBracket(); // 跳过 ')'
247.         return true;
248.     }
249.     return false;
250. }
251.
252. // 〈写语句〉 → write(〈表达式〉{, 〈表达式〉})
253. bool Parser::writeStatement()
254. {
255.     logs("LOG[Parser: writeStatement]");
256.     if (SYMBOL::WRITE == nowSymbolType)
257.     {
258.         /* 语义分析 */
259.         advance();                          // 跳过 'write'
260.         ASSERT(SYMBOL::LBR == nowSymbolType, EXCEPTION::MISSING_LBR); // 异常处理：缺少'('
261.         do
262.         {
263.             advance(); // 跳过 '('、','
264.             expression();
265.         } while (SYMBOL::COMMA == nowSymbolType);
266.         codeTable.push_back(new Code(CODE::OPR, OP::LINE)); // 输出 换行
267.         rightBracket(); // 跳过 ')'
268.         return true;
269.     }
270.     return false;
271. }
272.
273. // 〈复合语句〉 → begin〈语句〉{ ； 〈语句〉}〈end〉
274. bool Parser::beginStatement()
275. {
276.     logs("LOG[Parser: beginStatement]");
277.     if (SYMBOL::BEGIN == nowSymbolType)
278.     {
279.         do
280.         {
281.             advance();                      // 跳过 'begin' 或 ';'
282.             statement();
283.         } while (SYMBOL::SEMIC == nowSymbolType);
284.         ASSERT(SYMBOL::END == nowSymbolType, EXCEPTION::MISSING_END); // 异常处理：缺少
    'end'
285.         advance();                          // 跳过 'end'
286.         return true;
287.     }
288.     return false;
289. }
290.
291. // 〈分号〉
292. void Parser::semicolon()
293. {
294.     logs("LOG[Parser: semicolon]");
295.     ASSERT(SYMBOL::SEMIC == nowSymbolType, EXCEPTION::MISSING_SEMIC); // 异常处理：缺少';'
296.     advance();                              // 跳过 ';'
297. }
298.
299. void Parser::rightBracket()
300. {
301.     logs("LOG[Parser: rightBracket]");
302.     ASSERT(SYMBOL::RBR == nowSymbolType, EXCEPTION::MISSING_RBR); // 异常处理：缺少';'
```

```
303.     advance();                          // 跳过 ')'
304. }
305.
306. //  〈表达式〉 → [+|-]〈项〉{〈加减运算符〉 〈项〉}
307. //  〈加减运算符〉 → +|-
308. void Parser::expression()
309. {
310.     logs("LOG[Parser: expression]");
311.     SymbolType op = nowSymbolType;
312.     if (SYMBOL::ADD == nowSymbolType || SYMBOL::SUB == nowSymbolType)
313.         advance();          // 跳过 '+' / '-'
314.     term();
315.     while (SYMBOL::ADD == nowSymbolType || SYMBOL::SUB == nowSymbolType)
316.     {
317.         op = nowSymbolType;
318.         advance();              // 跳过 '+' / '-'
319.         term();
320.     }
321. }
322.
323. //  〈项〉 → 〈因子〉{〈乘除运算符〉 〈因子〉}
324. //  〈乘除运算符〉 → *|/
325. void Parser::term()
326. {
327.     logs("LOG[Parser: term]");
328.     factor();
329.     while (SYMBOL::MUL == nowSymbolType || SYMBOL::DIV == nowSymbolType)
330.     {
331.         advance(); // 跳过 '*' / '/'
332.         factor();
333.     }
334. }
335.
336. //  〈因子〉 → 〈标识符〉|〈无符号整数〉| (〈表达式〉)
337. void Parser::factor()
338. {
339.     logs("LOG[Parser: factor]");
340.     if (SYMBOL::IDENTIFIER == nowSymbolType)
341.     {
342.         std::string name = nowSymbol.getValue();
343.         advance();// 跳过标识符
344.         Symbol* symbol = symbolTable.getSymbol(name);
345.         ASSERT(nullptr != symbol, EXCEPTION::NEVER_DECLARE);  // 异常处理: 未声明
346.     } else if (SYMBOL::NUMBER == nowSymbolType)
347.     {
348.         advance(); // 跳过 '数字'
349.     } else if (SYMBOL::LBR == nowSymbolType)
350.     {
351.         advance(); // 跳过 '('
352.         expression();
353.         rightBracket();
354.     }
355. }
356.
357. //  〈条件〉 → 〈表达式〉〈关系运算符〉〈表达式〉| odd〈表达式〉
358. //  〈关系运算符〉 → =|#|<|<=|>|>=
359. void Parser::condition()
360. {
361.     logs("LOG[Parser: condition]");
362.     if (SYMBOL::ODD == nowSymbolType)
363.     {
364.         advance(); // 跳过 'odd'
365.         expression();
366.     }else
367.     {
368.         expression();
```

```
369.         SymbolType op = nowSymbolType; // 保存关系算符
370.         advance();                     // 跳过关系运算符
371.         expression();
372.     }
373. }
```

3.5 测试代码

在语法分析部分中，最主要的是生成了 TABLE 表，以及为语法树生成奠下基础，分别用“../test/PL0_code0.in”、…、“../test/PL0_code6.in”作为命令行启动参数，调用下面的 Parser 测试代码，可以测试语法分析器是否能正常工作。具体的输出结果，放置于附录中。

代码 3-8 Parser 测试代码

```
1.  #include <iostream>
2.  #include <fstream>
3.  #include "lexer.h"
4.  #include "parser.h"
5.
6.  /* 语法分析测试代码 */
7.  int main(int argc, char **argv)
8.  {
9.      std::ifstream fin(argv[1]);
10.     Lexer lexer(fin);
11.     Parser parser(lexer);
12.     parser.program();
13.     parser.printTables();
14.     return 0;
15. }
```

4 语法树生成

4.1 任务要求

打印一棵语法树，体现语法分析的规约过程。

4.2 语法树类的设计

语法树，首先是一棵树，需要设计一个节点类 SyntaxTreeNode，没有根据节点信息查找节点的需求，纯属是保存起节点的子节点。所以采用 list 作为容器，保存每个节点的子节点们，具体的声明代码如下：

代码 4-1 syntaxTree.h 中 SyntaxTreeNode 类

```
class SyntaxTreeNode
{
public:
    std::string name;
    std::list<SyntaxTreeNode*> children;
    explicit SyntaxTreeNode(std::string name): name(std::move(name)) {}
};
```

随后我们使用设计模式中的代理模式，创建一个 SyntaxTree 类，其要求持有根节点，以方便从根节点开始遍历，并且要维护一个语法树栈，因为递归下降分析法，是一种栈式的规约顺序，需要使用栈来完成规约（这里使用链表当作栈用，是因为 STL 的 stack 封装的太细致了，没法实现遍历），具体的声明代码如下：

代码 4-2 syntaxTree.h 中 SyntaxTree 类

```
class SyntaxTree
{
    std::list<SyntaxTreeNode*> syntaxTreeStack;  // 语法树 栈
    SyntaxTreeNode* root;                        // 语法树 根
public:
    void merge(Symbol* symbol);
    void merge(std::string name);
    void begin(const std::string &name);
    void end();
    void print();
    void print(SyntaxTreeNode* now, const std::string& prefix, bool isLast);
};
```

添加一个新的规约节点到栈中，需要调用“begin”函数，相应的，语法树节点规约结束后，需要调用“end”函数，在规约的过程中，简单 Symbol 规约到产生式，需要调用“merge”函数，具体的实现代码如下：

代码 4-3 syntaxTree.cpp 中部分代码

```
void SyntaxTree::begin(const std::string &name)
```

```
{
    SyntaxTreeNode *syntaxTreeNode = new SyntaxTreeNode(name);
    if (syntaxTreeStack.empty())
        this->root = syntaxTreeNode;
    else
        syntaxTreeStack.back()->children.push_back(syntaxTreeNode);
    syntaxTreeStack.push_back(syntaxTreeNode);
}

void SyntaxTree::end()
{
    if (syntaxTreeStack.back()->children.empty())
    {
        syntaxTreeStack.pop_back();
        syntaxTreeStack.back()->children.pop_back();
    } else
    {
        syntaxTreeStack.pop_back();
    }
}

void SyntaxTree::merge(Symbol *symbol)
{
    syntaxTreeStack.back()->children.push_back(new SyntaxTreeNode(symbol->getValue()));
}

void SyntaxTree::merge(std::string name)
{
    syntaxTreeStack.back()->children.push_back(new SyntaxTreeNode(std::move(name)));
}
```

对于大规模规约的语法树，横向打印难度大且观看不便。思考了几种方案，最后决定采用类似“tree”指令的 style 来打印语法树，只需要设计好递归程序的传参和执行即可，并且父子节点间关系明了。

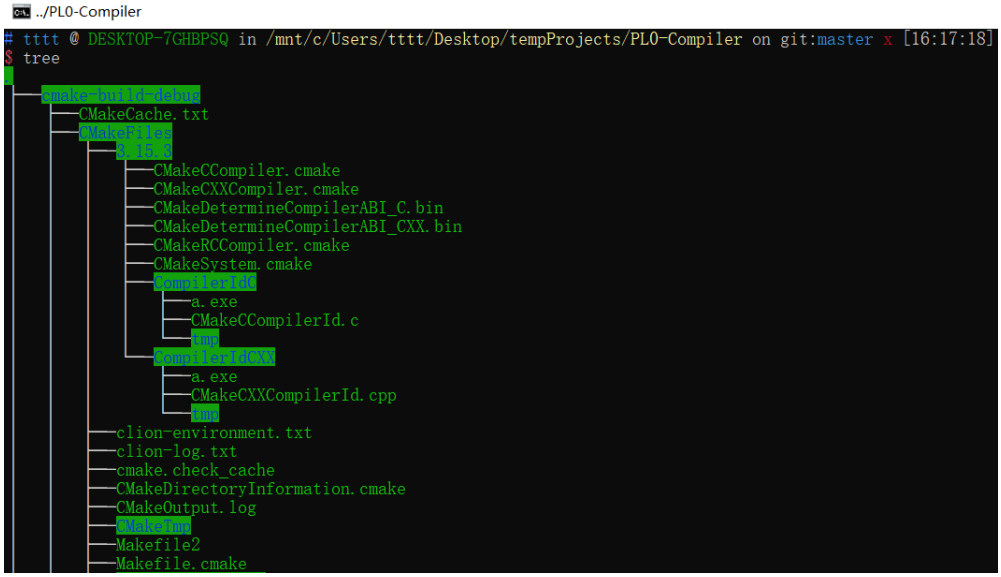

图 4-1 命令行中 tree 指令的执行效果

具体的语法树打印实现代码如下：

代码 4-4 syntaxTree.cpp 中打印语法树部分的代码

```
void SyntaxTree::print()
{
    std::cout << root->name << std::endl;
```

```
    int n = root->children.size();
    for (auto x : root->children)
        print(x, "", --n==0);
}

void SyntaxTree::print(SyntaxTreeNode *now, const std::string &prefix, bool isLast)
{
    std::cout << prefix;
    std::cout << (isLast ? "|__" : "|--");
    std::cout << now->name << std::endl;

    int n = now->children.size();
    for (auto x : now->children)
        print(x, prefix + (isLast ? "    " : "|   "), --n == 0);
}
```

4.3 语法分析器改动部分

首先对于产生式，必须得调用“begin”和“end”方法，由于 C++没有动态代理可以对方法做增强，所以这里只能显式地编写调用“begin”的代码，举例如下，对于所有的产生式规约子程序，都必须在头尾显式地如此编写调用“begin”和“end”的方法，篇幅有限正文部分不多列举：

代码 4-5　parser.cpp 中规约产生式<程序>的代码

```
// 〈程序〉→〈分程序〉.
void Parser::program()
{
    logs("LOG[Parser: program]");
    syntaxTree.begin("[program]");

    /* 语义分析 */
    subProbgram();                          // 〈分程序〉
    ASSERT(SYMBOL::POINT == nowSymbolType, EXCEPTION::MISSING_SEMICOLON); // 异常处理: 缺少 '.'
    advance();                              // 跳过 '.'
    ASSERT(lexer.isEOF(), EXCEPTION::EXTRA_CHARACTERS); // 异常处理: 文件尾 '.' 后还有字符

    syntaxTree.end();
}
```

随后是规约到产生式的简单 Symbol，由于所有 Symbol 的递进都要经过“advance”函数，所以在可以改造“advance”以方便 Symbol 可以规约 merge 到具体的产生式节点上，具体的修改代码如下：

代码 4-6　parser.cpp 中 advance 函数

```
// 取下一个 词
void Parser::advance()
{
    if (SYMBOL::ILLEGAL != nowSymbolType)
    {
        if (!nowSymbol.getValue().empty())
            syntaxTree.merge(nowSymbol.getValue());
        else if(-1 != nowSymbol.getNumber())
            syntaxTree.merge(std::to_string(nowSymbol.getNumber()));
        else
            syntaxTree.merge(nowSymbolType.name);
```

```
    }
    nowSymbol = lexer.getSymbol();
    nowSymbolType = nowSymbol.getSymbolType();
    logs("LOG[Parser: advance] ", nowSymbol);
}
```

4.4 测试代码

在语法分析的部分，为语法树生成奠下基础，语法树生成模块类似于语法分析器的装饰器，需要更改的代码不多。测试部分，分别用“../test/PL0_code0.in”、…、“../test/PL0_code6.in”作为命令行启动参数，调用下面的 SyntaxTree 测试代码，可以测试语法树输出是否能正常工作。具体的输出结果，放置于附录中。

代码 4-7　SyntaxTree 测试代码

```
#include <iostream>
#include <fstream>
#include "lexer.h"
#include "parser.h"
/* 语法树生成测试代码 */
int main(int argc, char **argv)
{
    std::ifstream fin(argv[1]);
    Lexer lexer(fin);
    Parser parser(lexer);
    parser.program();
    parser.syntaxTree.print();
    return 0;
}
```

5 目标代码生成

5.1 任务要求

语句处理和代码生成：对语句逐句分析，语法正确则生目标代码，当遇到标识符的引用则去查 TABLE 表，看是否有过正确的定义，若有则从表中取出相关的信息，供代码生成用。PL/0 语言的代码生成是由过程 GEN 完成。GEN 过程有三个参数，分别代表目标代码的功能码、层差、和位移量。生成的目标代码放在数组 CODE 中。CODE 是一维数组，数组元素是结构体类型数据。

PL/0 语言的目标指令是一种假想的栈式计算机的汇编语言，其格式为 [f, l, a]，其中 f 代表功能码，l 代表层次差，a 代表位移量。目标指令有 8 条：

① LIT：将常数放到运栈顶，a 域为常数。

② LOD：将变量放到栈顶。a 域为变量在所说明层中的相对位置，l 为调用层与说明层的层差值。

③ STO：将栈顶的内容送到某变量单元中。a 域为变量在所说明层中的相对位置，l 为调用层与说明层的层差值。

④ CAL：调用过程的指令。a 为被调用过程的目标程序的入口地址，l 为层差。

⑤ INT：为被调用的过程（或主程序）在运行栈中开辟数据区。a 域为开辟的个数。

⑥ JMP：无条件转移指令，a 为转向地址。

⑦ JPC：条件转移指令，当栈顶的布尔值为非真时，转向 a 域的地址，否则顺序执行。

⑧ OPR：关系和算术运算。具体操作由 a 域给出。运算对象为栈顶和次顶的内容进行运算，结果存放在次顶。a 域为 0 时是退出数据区。

5.2 目标代码类的设计

目标代码一共有 8 类，可以为之设计枚举类，枚举类作为强类型对编码更加友好。同时声明 map 常量便于将目标代码类型与具体的字符串联系起来，外部使用命名空间包装，以防止命名空间污染，声明代码如下：

代码 5-1 code.h 中目标代码枚举类

```
namespace CODE
{
    enum CODE_TYPE
    {
        LIT, // 将常数放到运栈顶，a 域为常数
        LOD, // 将变量放到栈顶。a 域为变量在所说明层中的相对位置，l 为调用层与说明层的层差值
        STO, // 将栈顶的内容送到某变量单元中。a 域为变量在所说明层中的相对位置，l 为调用层与说明层的层差值
        CAL, // 调用过程的指令。a 为被调用过程的目标程序的入口地址，l 为层差
        INT, // 为被调用的过程（或主程序）在运行栈中开辟数据区。a 域为开辟的个数
        JMP, // 无条件转移指令，a 为转向地址
        JPC, // 条件转移指令，当栈顶的布尔值为非真时，转向 a 域的地址，否则顺序执行
```

```
        OPR  // 关系和算术运算。具体操作由 a 域给出。运算对象为栈顶和次顶的内容进行运算，结果存放在次顶。a 域为 0 时是退出数据区
    };
    const std::unordered_map<int, std::string> CODE_TYPE_MAP({
        {LIT, "LIT"},
        {LOD, "LOD"},
        {STO, "STO"},
        {CAL, "CAL"},
        {INT, "INT"},
        {JMP, "JMP"},
        {JPC, "JPC"},
        {OPR, "OPR"}
    });
}
```

对于“OPR”类型的目标代码，具体的操作由“a”域给出，采用同样的思想，对“OPR”的“a”域也采用相似的枚举类设计，具体的声明代码如下：

代码 5-2　code.h 中 OPR 操作码枚举类

```
namespace OP
{
    enum OP_TYPE
    {
        RET = 0,   // 过程调用结束后，返回调用点并退栈
        NEG = 1,   // 栈顶元素取反
        ADD = 2,   // 弹出次栈顶与栈顶相加，结果进栈
        SUB = 3,   // 弹出次栈顶减去栈顶，结果进栈
        MUL = 4,   // 弹出次栈顶、栈顶，相乘结果进栈
        DIV = 5,   // 弹出次栈顶除以栈顶，结果值进栈
        LEQ = 6,   // 栈顶两个元素弹出，判次栈顶小于等于栈顶结果进栈
        LS = 7,    // 栈顶两个元素弹出，判次栈顶小于栈顶结果进栈
        EQU = 8,   // 弹出栈顶两元素判相等结果入栈顶
        NEQ = 9,   // 弹出栈顶两元素判不等结果入栈顶
        GT = 10,   // 栈顶两个元素弹出，判次栈顶大于栈顶结果进栈
        GEQ = 11,  // 栈顶两个元素弹出，判次栈顶小于等于栈顶结果进栈
        WRITE = 14,// 栈顶值输出至屏幕
        LINE = 15, // 屏幕输出换行
        READ = 16, // 从命令行读一个数到栈顶
        ODD = 17,  // 弹出栈顶，判断是否为奇数，结果进栈
    };
    const std::unordered_map<int, std::string> OP_TYPE_MAP({
        {RET, "RET"},
        {NEG, "NEG"},
        {ADD, "ADD"},
        {SUB, "SUB"},
        {MUL, "MUL"},
        {DIV, "DIV"},
        {LEQ, "LEQ"},
        {LS, "LS"},
        {EQU, "EQU"},
        {NEQ, "NEQ"},
        {GT, "GT"},
        {GEQ, "GEQ"},
        {WRITE, "WRITE"},
        {LINE, "LINE"},
        {READ, "READ"}
    });
}
```

目标代码是三地址代码，设计“Code”类来承载具体的三地址信息，同时使用 getter、setter 来封装数据接口，重写输出流方法便于输出，具体的声明代码如下：

代码 5-3　code.h 中 Code 类

```
class Code
{
private:
    CODE::CODE_TYPE f;
    int l = 0;
    int a = 0;
public:
    void setL(int _l)
    {
        this->l = _l;
    }

    void setA(int _a)
    {
        this->a = _a;
    }

    explicit Code(CODE::CODE_TYPE _f) : f(_f)
    {}

    Code(CODE::CODE_TYPE _f, int _l, int _a) : f(_f), l(_l), a(_a)
    {}

    Code(CODE::CODE_TYPE _f, int _a) : f(_f), a(_a)
    {}

    friend std::ostream &operator<<(std::ostream &os, const Code &code)
    {
        os << CODE::CODE_TYPE_MAP.find(code.f)->second << "\t" << code.l << "\t" << code.a;
        if (CODE::OPR == code.f)
            os << '(' << OP::OP_TYPE_MAP.find(code.a)->second << ')';
        return os;
    }

    CODE::CODE_TYPE getF() const
    {
        return f;
    }

    int getL() const
    {
        return l;
    }

    int getA() const
    {
        return a;
    }
};
```

5.3　语法分析器的改动部分

完成必要的目标代码相关类设计后，需要改造现有的语法分析器，使得在语法分析的同时生成目标代码，语法分析和目标代码生成只需要扫描一遍代码即可完成。

对于语法分析器的改造，本质上是对具体语句规约动作的改造，在规约的同时产生目标代码，例如在“if …then…”语句规约时，产生了条件转移代码，并且待执行语句的语义分析完产生具体的执行代码后，回填条件转移代码的地址，对“bool Parser::ifStatement()”改造如

下：

代码 5-4　parser.cpp 中“bool Parser::ifStatement()”改造

```
// 〈条件语句〉 → if〈条件〉then〈语句〉
bool Parser::ifStatement()
{
    logs("LOG[Parser: ifStatement]");
    if (SYMBOL::IF == nowSymbolType)
    {
        syntaxTree.begin("[ifStatement]");

        /* 语义分析 */
        advance();                          // 跳过 'IF'
        condition();
        ASSERT(SYMBOL::THEN == nowSymbolType, EXCEPTION::MISSING_THEN); // 缺少 THEN
        advance();                          // 跳过 'THEN'

        /* 中间代码生成 */
        Code *jpc = new Code(CODE::JPC);// 条件转移代码，地址回填
        codeTable.push_back(jpc);

        /* 语义分析 */
        statement();

        /* 中间代码生成 */
        jpc->setA(getCodeAddress());   // 回填地址

        syntaxTree.end();
        return true;
    }
    return false;
}
```

另一个例子，对于表达式的规约来说，可能会产生运算，运算是将栈顶的若干数据执行运算后放入新结果，表达式定义的语义里有“+”和“-”两种运算，项定义的语义里有“*”和“/”两种运算，对于它们的改造就涉及生成“OPR”类型的中间代码：

代码 5-5　parser.cpp 中规约表达式和项部分的改造

```
// 〈表达式〉 → [+|-]〈项〉{〈加减运算符〉〈项〉}
// 〈加减运算符〉 → +|-
void Parser::expression()
{
    logs("LOG[Parser: expression]");
    syntaxTree.begin("[expression]");

    SymbolType op = nowSymbolType;
    if (SYMBOL::ADD == nowSymbolType || SYMBOL::SUB == nowSymbolType)
        advance();          // 跳过 '+' / '-'
    term();
    if (SYMBOL::SUB == op) // 栈顶 取负
        codeTable.push_back(new Code(CODE::OPR,  OP::NEG));
    while (SYMBOL::ADD == nowSymbolType || SYMBOL::SUB == nowSymbolType)
    {
        op = nowSymbolType;
        advance();              // 跳过 '+' / '-'
        term();
        if (SYMBOL::ADD == op) // 栈顶两个元素弹出相加结果进栈
            codeTable.push_back(new Code(CODE::OPR, OP::ADD));
        else                    // 栈顶两个元素弹出，次栈顶减去栈顶结果进栈
            codeTable.push_back(new Code(CODE::OPR, OP::SUB));
    }
```

```
24.
25.     syntaxTree.end();
26. }
27.
28. // 〈项〉 → 〈因子〉{〈乘除运算符〉〈因子〉}
29. // 〈乘除运算符〉 → *|/
30. void Parser::term()
31. {
32.     logs("LOG[Parser: term]");
33.     syntaxTree.begin("[term]");
34.
35.     factor();
36.     while (SYMBOL::MUL == nowSymbolType || SYMBOL::DIV == nowSymbolType)
37.     {
38.         SymbolType op = nowSymbolType;
39.         advance(); // 跳过 '*' / '/'
40.         factor();
41.         if (SYMBOL::MUL == op) // 栈顶两个元素弹出相乘结果进栈
42.             codeTable.push_back(new Code(CODE::OPR, OP::MUL));
43.         else                   // 栈顶两个元素弹出，次栈顶除以栈顶结果进栈
44.             codeTable.push_back(new Code(CODE::OPR, OP::DIV));
45.     }
46.
47.     syntaxTree.end();
48. }
```

5.4 测试代码

在语法分析的部分，为目标代码生成奠下基础，尽管目标代码生成的改动部分需要侵入原本的语法分析代码，但是加注释后区分好即可便于区分和维护，总体来说需要更改的代码不多。测试部分，分别用“../test/PL0_code0.in”、…、“../test/PL0_code6.in”作为命令行启动参数，调用下面的 Code 生成测试代码，可以测试中间代码生成是否能正常工作。具体的输出结果，放置于附录中。

代码 5-6　Code 生成测试代码

```
1.  #include <iostream>
2.  #include <fstream>
3.  #include "lexer.h"
4.  #include "parser.h"
5.
6.  /* 目标代码生成测试代码 */
7.  int main(int argc, char **argv)
8.  {
9.      std::ifstream fin(argv[1]);
10.     Lexer lexer(fin);
11.     Parser parser(lexer);
12.     parser.program();
13.     parser.printCode();
14.     return 0;
15. }
```

6 解释执行器

6.1 任务要求

编译结束后，记录源程序中标识符的 TABLE 表已退出内存，内存中只剩下用于存放目标程序的 CODE 数组和运行时的数据区 S。S 是由解释程序定义的一维整型数组。解释执行时的数据空间 S 为栈式计算机的存储空间。遵循后进先出的规则，对每个过程（包括主程序）当被调用时，才分配数据空间，退出过程时，则所分配的数据空间被释放。

为解释程序定义四个寄存器：

（1）I：指令寄存器，存放当前正在解释的一条目标指令。

（2）P：程序地址寄存器，指向下一条要执行的目标指令（相当于 CODE 数组的下标）。

（3）T：栈顶寄存器，每个过程运行时要为它分配数据区（或称为数据段），该数据区分为两部分。

静态部分：包括变量存放区和三个联单元。

动态部分：作为临时工作单元和累加器用。需要时临时分配，用完立即释放。栈顶寄存器 T 指出了当前栈中最新分配的单元（T 也是数组 S 的下标）。

（4）B：基地址寄存器，指出每个过程被调用时，在数据区 S 中给出它分配的数据段起始地址，也称为基地址。每个过程被调用时，在栈顶分配三个联系单元。这三个单元的内容分别是：

SL：静态链，它是指向定义该过程的直接外过程运行时数据段的基地址。

DL：动态链，它是指向调用该过程前正在运行过程的数据段的基地址。

RA：返回地址，记录调用该过程时目标程序的断点，即当时的程序地址寄存器 P 的值。

具体的过程调用和结束，对上述寄存器及三个联系单元的填写和恢复由下列目标指令完成。

“INT 0 a”。a 为局部量个数加 3

“OPR 0 0”。恢复调用该过程前正在运行过程（或主程序）的数据段的基地址寄存器的值，恢复栈顶寄存器 T 的值，并将返回地址送到指令寄存器 P 中。

“CAL l a”。a 为被调用过程的目标程序的入口，送入指令地址寄存器 P 中。CAL 指令还完成填写静态链，动态链，返回地址，给出被调用过程的基地址值，送入基址寄存器 B 中。

6.2 执行器的设计

执行器设计部分，与操作系统课设中 MIPS 代码的虚拟机模拟执行类似，维护一个栈代表数据区，维护一个数组代表代码区，特判每一条代码的类型，为之设计执行逻辑，写成 C++ 代码去模拟执行，与操作系统课设中写 MIPS 虚拟机类似。所以设计一个“VM”类，输入是

中间代码数组，即代码区，可以启动执行。除了数据区的栈外，需要维护栈顶指针 top，基地址 base 等，具体的声明代码如下：

代码 6-1 “vm.h”代码

```
/*
 * Copyright 2020 ZhangT. All Rights Reserved.
 * Author: ZhangT
 * Author-Github: github.com/zhangt2333
 * vm.h 2020/5/18
 * Comments:
 */
#ifndef PL0_COMPILER_VM_H
#define PL0_COMPILER_VM_H


#include <utility>
#include <vector>
#include "code.h"

class VM
{
public:
    VM(std::vector<Code *> codeTable) : codeTable(std::move(codeTable))
    {}
    void run();

private:
    std::vector<Code*> codeTable;
    int stack[1024];
    int top;
    int base;
    int pc;
    Code* code;
    int getBaseAddress(int nowBaseAddress, int levelDiff);
    void opr();
};


#endif //PL0_COMPILER_VM_H
```

具体的实现部分，需要注意的点如下，首先是根据层差获取基地址，通过动态链不断向外找到调用层基地址，其次，在“run”的设计中，采用“switch”语句特判代码类型，针对不同的代码特别地设计执行逻辑，另外将“OPR”单独拎出来作为一个方法，便于理清逻辑，减少单个方法的代码量。具体实现如下：

代码 6-2 “vm.cpp”代码

```
/*
 * Copyright 2020 ZhangT. All Rights Reserved.
 * Author: ZhangT
 * Author-Github: github.com/zhangt2333
 * vm.cpp 2020/5/18
 * Comments:
 */
#include <iostream>
#include <cstring>
```

```
10. #include "vm.h"
11. 
12. 
13. int VM::getBaseAddress(int nowBaseAddress, int levelDiff)
14. {
15.     while (levelDiff--)
16.         nowBaseAddress = stack[nowBaseAddress + 1];
17.     return nowBaseAddress;
18. }
19. 
20. void VM::run()
21. {
22.     top = pc = base = 0;
23.     memset(stack, 0, sizeof(stack));
24.     do
25.     {
26.         code = codeTable.at(pc++);
27.         switch (code->getF())
28.         {
29.             // 常量送栈顶
30.             case CODE::LIT:
31.                 stack[top++] = code->getA();
32.                 break;
33.             // 变量送栈顶
34.             case CODE::LOD:
35.                 stack[top++] = stack[getBaseAddress(base, code->getL()) + code->getA()];
36.                 break;
37.             // 栈顶送变量
38.             case CODE::STO:
39.                 stack[getBaseAddress(base, code->getL()) + code->getA()] = stack[--top];
40.                 break;
41.             // 调用
42.             case CODE::CAL:
43.                 stack[top] = base;   // 静态连
44.                 stack[top + 1] = getBaseAddress(base, code->getL()); // 动态链
45.                 stack[top + 2] = pc; // 返回地址
46.                 base = top;          // 不修改 top，前面已将 address+3，生成 code 后会产生 INT 语句，
    修改 top 值
47.                 pc = code->getA();
48.                 break;
49.             // 开辟空间
50.             case CODE::INT:
51.                 top = top + code->getA(); // 3(静态链、动态链、返回地址)+变量+常量
52.                 break;
```

```
53.             // 无条件跳转
54.             case CODE::JMP:
55.                 pc = code->getA();
56.                 break;
57.             // 有条件跳转
58.             case CODE::JPC:
59.                 if (stack[top - 1] == 0)
60.                     pc = code->getA();
61.                 break;
62.             // 操作
63.             case CODE::OPR:
64.                 this->opr();
65.                 break;
66.         }
67.     } while (pc != 0);
68. }
69.
70. void VM::opr()
71. {
72.     switch (code->getA())
73.     {
74.         case OP::RET:
75.             top = base;
76.             pc = stack[base + 2];// 返回地址
77.             base = stack[base];  // 静态连
78.             break;
79.         case OP::NEG:
80.             stack[top - 1] = -stack[top - 1];
81.             break;
82.         case OP::ADD:
83.             stack[top - 2] = stack[top - 1] + stack[top - 2];
84.             --top;
85.             break;
86.         case OP::SUB:
87.             stack[top - 2] = stack[top - 2] - stack[top - 1];
88.             --top;
89.             break;
90.         case OP::MUL:
91.             stack[top - 2] = stack[top - 2] * stack[top - 1];
92.             --top;
93.             break;
94.         case OP::DIV:
95.             stack[top - 2] = stack[top - 2] / stack[top - 1];
96.             --top;
```

```
97.             break;
98.         case OP::LEQ:
99.             stack[top - 2] = stack[top - 2] <= stack[top - 1];
100.            --top;
101.            break;
102.        case OP::LS:
103.            stack[top - 2] = stack[top - 2] < stack[top - 1];
104.            --top;
105.            break;
106.        case OP::EQU:
107.            stack[top - 2] = stack[top - 2] == stack[top - 1];
108.            --top;
109.            break;
110.        case OP::NEQ:
111.            stack[top - 2] = stack[top - 2] != stack[top - 1];
112.            --top;
113.            break;
114.        case OP::GT:
115.            stack[top - 2] = stack[top - 2] > stack[top - 1];
116.            --top;
117.            break;
118.        case OP::GEQ:
119.            stack[top - 2] = stack[top - 2] >= stack[top - 1];
120.            --top;
121.            break;
122.        case OP::WRITE:
123.            std::cout << stack[top - 1] << '\t';
124.            break;
125.        case OP::LINE:
126.            std::cout << std::endl;
127.            break;
128.        case OP::READ:
129.            std::cout << "Please Input: ";
130.            std::cin >> stack[top++];
131.            break;
132.        case OP::ODD:
133.            stack[top - 1] &= 1;
134.            break;
135.    }
136. }
```

6.3 测试代码

解释执行和前面的语法分析等部分都是独立的，体现了该编译系统的解耦性。解释执行器是一个单独的类，可以独立运行，只需要输入中间代码表。测试部分，分别用“../test/PL0_code0.in”、…、“../test/PL0_code6.in”作为命令行启动参数，调用下面的解释执行测试代码，可以测试中间代码生成是否能正常工作。具体的输出结果，放置于附录中。

代码 6-3　解释执行测试代码

```
#include <iostream>
#include <fstream>
#include "lexer.h"
#include "parser.h"
#include "vm.h"

/* 解释执行测试代码 */
int main(int argc, char **argv)
{
    std::ifstream fin(argv[1]);
    Lexer lexer(fin);
    Parser parser(lexer);
    parser.program();
    VM vm(parser.codeTable);
    vm.run();
    return 0;
}
```

7 附录

7.1 测试样例代码

7.1.1 PL0_code0.in

代码 7-1 PL0_code0.in

```
const a=10;
var d,e,f;
procedure p;
var g;
begin
  d:=a*2;
  e:=a/3;
  if d<=e then f:=d+e
end;
begin
  read(e,f);
  write(e,f,d);
  call p;
  while odd d do e:=-e+1
end.
```

7.1.2 PL0_code1.in

代码 7-2 PL0_code1.in

```
const a=10;
var b,c;
procedure p;
begin
  c:=b+a
end;
begin
  read(b);
  while b#0 do
    begin
      call  p;
      write(2*c);
      read(b);
    end
end.
```

7.1.3 PL0_code2.in

代码 7-3 PL0_code2.in

```
var x, squ;

procedure square;
begin
   squ:= x * x
end;

```

```
8.  begin
9.     x := 1;
10.    while x <= 10 do
11.    begin
12.       call square;
13.       write(squ);
14.       x := x + 1
15.    end
16. end.
```

7.1.4 PL0_code3.in

代码 7-4 PL0_code3.in

```
1.  const max = 100;
2.  var arg, ret;
3.
4.  procedure isprime;
5.  var i;
6.  begin
7.      ret := 1;
8.      i := 2;
9.      while i < arg do
10.     begin
11.         if arg / i * i = arg then
12.         begin
13.             ret := 0;
14.             i := arg
15.         end;
16.         i := i + 1
17.     end
18. end;
19.
20. procedure primes;
21. begin
22.     arg := 2;
23.     while arg < max do
24.     begin
25.         call isprime;
26.         if ret = 1 then write (arg);
27.         arg := arg + 1
28.     end
29. end;
30.
31. call primes
32. .
```

7.1.5 PL0_code4.in

代码 7-5 PL0_code4.in

```
1.  var x, y, z, q, r, n, f;
2.
3.  procedure multiply;
4.  var a, b;
5.  begin
6.    a := x;
7.    b := y;
8.    z := x*y
9.  end;
10.
11. procedure divide;
```

```
var w;
begin
  r := x;
  q := 0;
  w := y;
  while w <= r do w := 2 * w;
  while w > y do
  begin
    q := 2 * q;
    w := w / 2;
    if w <= r then
    begin
      r := r - w;
      q := q + 1
    end
  end
end;

procedure gcd;
var  g;
begin
  f := x;
  g := y;
  while f # g do
  begin
    if f < g then g := g - f;
    if g < f then f := f - g
  end;
  z := f
end;

procedure fact;
begin
  if n > 1 then
  begin
    f := n * f;
    n := n - 1;
    call fact
  end
end;

begin
  read(x);read(y); call multiply;write(z);
  read(x);read(y); call divide; write(q); write(r);
  read(x);read(y); call gcd; write(z);
  read(n); f := 1; call fact; write(f)
end.
```

7.1.6 PL0_code5.in

代码 7-6 PL0_code5.in

```
const c1=2;
var v1,v2,v3,v4;
procedure p1;
 var v5;
 begin
  v5:=2;
  write(v5/2+2-1);
  while v3#0 do
  begin
   v4:=v1/v2;
   v3:=v1-v4*v2;
   v1:=v2;
```

```
    v2:=v3;
   end;
  end;
 procedure p2;
  const c2=2;
  procedure p3;
   begin
    if v1#1 then
    begin
     v1:=v1-1;
     v2:=v2*v1;
     call p3;
    end;
   end;
  begin
   call p3;
   if odd c2 then
    write(c2);
   if c2=2 then
    write(c2+1)
  end;
 begin
  read(v1,v2);
  if v1<v2 then
  begin
   v3:=v1;
   v1:=v2;
   v2:=v3;
  end;
  begin;
   v3:=1;
   call p1;
   write(c1,c1*v1,c1*v1/2);
  end;
  read(v1);
  v2:=v1;
  call p2;
  write(v2);
 end.
```

7.2 词法分析结果

7.2.1 PL0_code0.in 词法分析结果

代码 7-7 PL0_code0.in 词法分析结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code0.in
const identifier(a) = number(10) ;
var identifier(d) , identifier(e) , identifier(f) ;
procedure identifier(p) ;
var identifier(g) ;
begin identifier(d) := identifier(a) * number(2) ;
identifier(e) := identifier(a) / number(3) ;
if identifier(d) <= identifier(e) then identifier(f) := identifier(d) + identifier(e) end ;
begin read ( identifier(e) , identifier(f) ) ;
write ( identifier(e) , identifier(f) , identifier(d) ) ;
call identifier(p) ;
while odd identifier(d) do identifier(e) := - identifier(e) + number(1) end .
Process finished with exit code 0
```

7.2.2 PL0_code1.in 词法分析结果

代码 7-8 PL0_code1.in 词法分析结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code1.in
const identifier(a) = number(10) ;
var identifier(b) , identifier(c) ;
procedure identifier(p) ;
begin identifier(c) := identifier(b) + identifier(a) end ;
begin read ( identifier(b) ) ;
while identifier(b) # number(0) do begin call identifier(p) ;
write ( number(2) * identifier(c) ) ;
read ( identifier(b) ) ;
end end .
Process finished with exit code 0
```

7.2.3 PL0_code2.in 词法分析结果

代码 7-9 PL0_code2.in 词法分析结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code2.in
var identifier(x) , identifier(squ) ;
procedure identifier(square) ;
begin identifier(squ) := identifier(x) * identifier(x) end ;
begin identifier(x) := number(1) ;
while identifier(x) <= number(10) do begin call identifier(square) ;
write ( identifier(squ) ) ;
identifier(x) := identifier(x) + number(1) end end .
Process finished with exit code 0
```

7.2.4 PL0_code3.in 词法分析结果

代码 7-10 PL0_code3.in 词法分析结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code3.in
const identifier(max) = number(100) ;
var identifier(arg) , identifier(ret) ;
procedure identifier(isprime) ;
var identifier(i) ;
begin identifier(ret) := number(1) ;
identifier(i) := number(2) ;
while identifier(i) < identifier(arg) do begin if identifier(arg) / identifier(i) * identifier(i) = identifier(arg) then
 begin identifier(ret) := number(0) ;
identifier(i) := identifier(arg) end ;
identifier(i) := identifier(i) + number(1) end end ;
procedure identifier(primes) ;
begin identifier(arg) := number(2) ;
while identifier(arg) < identifier(max) do begin call identifier(isprime) ;
if identifier(ret) = number(1) then write ( identifier(arg) ) ;
identifier(arg) := identifier(arg) + number(1) end end ;
call identifier(primes) .
Process finished with exit code 0
```

7.2.5 PL0_code4.in 词法分析结果

代码 7-11 PL0_code4.in 词法分析结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code4.in
var identifier(x) , identifier(y) , identifier(z) , identifier(q) , identifier(r) , identifier(n) , identifier(f) ;
procedure identifier(multiply) ;
var identifier(a) , identifier(b) ;
begin identifier(a) := identifier(x) ;
identifier(b) := identifier(y) ;
identifier(z) := identifier(x) * identifier(y) end ;
procedure identifier(divide) ;
var identifier(w) ;
begin identifier(r) := identifier(x) ;
identifier(q) := number(0) ;
identifier(w) := identifier(y) ;
while identifier(w) <= identifier(r) do identifier(w) := number(2) * identifier(w) ;
while identifier(w) > identifier(y) do begin identifier(q) := number(2) * identifier(q) ;
identifier(w) := identifier(w) / number(2) ;
if identifier(w) <= identifier(r) then begin identifier(r) := identifier(r) - identifier(w) ;
identifier(q) := identifier(q) + number(1) end end end ;
procedure identifier(gcd) ;
var identifier(g) ;
begin identifier(f) := identifier(x) ;
identifier(g) := identifier(y) ;
while identifier(f) # identifier(g) do begin if identifier(f) < identifier(g) then identifier(g) := identifier(g) - iden
tifier(f) ;
if identifier(g) < identifier(f) then identifier(f) := identifier(f) - identifier(g) end ;
identifier(z) := identifier(f) end ;
procedure identifier(fact) ;
begin if identifier(n) > number(1) then begin identifier(f) := identifier(n) * identifier(f) ;
identifier(n) := identifier(n) - number(1) ;
call identifier(fact) end end ;
begin read ( identifier(x) ) ;
read ( identifier(y) ) ;
call identifier(multiply) ;
write ( identifier(z) ) ;
read ( identifier(x) ) ;
read ( identifier(y) ) ;
call identifier(divide) ;
write ( identifier(q) ) ;
write ( identifier(r) ) ;
read ( identifier(x) ) ;
read ( identifier(y) ) ;
call identifier(gcd) ;
write ( identifier(z) ) ;
read ( identifier(n) ) ;
identifier(f) := number(1) ;
call identifier(fact) ;
write ( identifier(f) ) end .
Process finished with exit code 0
```

7.2.6 PL0_code5.in 词法分析结果

代码 7-12 PL0_code5.in 词法分析结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code5.in
const identifier(c1) = number(2) ;
var identifier(v1) , identifier(v2) , identifier(v3) , identifier(v4) ;
procedure identifier(p1) ;
var identifier(v5) ;
begin identifier(v5) := number(2) ;
write ( identifier(v5) / number(2) + number(2) - number(1) ) ;
while identifier(v3) # number(0) do begin identifier(v4) := identifier(v1) / identifier(v2) ;
identifier(v3) := identifier(v1) - identifier(v4) * identifier(v2) ;
identifier(v1) := identifier(v2) ;
identifier(v2) := identifier(v3) ;
end ;
end ;
procedure identifier(p2) ;
const identifier(c2) = number(2) ;
procedure identifier(p3) ;
begin if identifier(v1) # number(1) then begin identifier(v1) := identifier(v1) - number(1) ;
identifier(v2) := identifier(v2) * identifier(v1) ;
call identifier(p3) ;
end ;
end ;
begin call identifier(p3) ;
if odd identifier(c2) then write ( identifier(c2) ) ;
if identifier(c2) = number(2) then write ( identifier(c2) + number(1) ) end ;
begin read ( identifier(v1) , identifier(v2) ) ;
if identifier(v1) < identifier(v2) then begin identifier(v3) := identifier(v1) ;
identifier(v1) := identifier(v2) ;
identifier(v2) := identifier(v3) ;
end ;
begin ;
identifier(v3) := number(1) ;
call identifier(p1) ;
write ( identifier(c1) , identifier(c1) * identifier(v1) , identifier(c1) * identifier(v1) / number(2) ) ;
end ;
read ( identifier(v1) ) ;
identifier(v2) := identifier(v1) ;
call identifier(p2) ;
write ( identifier(v2) ) ;
end .
Process finished with exit code 0
```

7.3 语法分析结果

7.3.1 PL0_code0.in 语法分析结果

代码 7-13 PL0_code0.in 语法分析结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code0.in
TX0->
NAME: a         KIND: const     VAL: 10
NAME: d         KIND: var       LEVEL: 0         ADR: 3
NAME: e         KIND: var       LEVEL: 0         ADR: 4
NAME: f         KIND: var       LEVEL: 0         ADR: 5
NAME: p         KIND: procedure LEVEL: 0         ADR: 2
TX1->
NAME: g         KIND: var       LEVEL: 1         ADR: 3
```

```
10.
11. Process finished with exit code 0
```

7.3.2 PL0_code1.in 语法分析结果

代码 7-14 PL0_code1.in 语法分析结果

```
1.  C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\
    TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code1.in
2.  TX0->
3.  NAME: a         KIND: const     VAL: 10
4.  NAME: b         KIND: var       LEVEL: 0         ADR: 3
5.  NAME: c         KIND: var       LEVEL: 0         ADR: 4
6.  NAME: p         KIND: procedure LEVEL: 0         ADR: 2
7.  TX1->
8.
9.  Process finished with exit code 0
```

7.3.3 PL0_code2.in 语法分析结果

代码 7-15 PL0_code2.in 语法分析结果

```
1.  C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\
    TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code2.in
2.  TX0->
3.  NAME: x         KIND: var       LEVEL: 0         ADR: 3
4.  NAME: squ       KIND: var       LEVEL: 0         ADR: 4
5.  NAME: square    KIND: procedure LEVEL: 0         ADR: 2
6.  TX1->
7.
8.  Process finished with exit code 0
```

7.3.4 PL0_code3.in 语法分析结果

代码 7-16 PL0_code3.in 语法分析结果

```
1.  C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\
    TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code3.in
2.  TX0->
3.  NAME: max       KIND: const     VAL: 100
4.  NAME: arg       KIND: var       LEVEL: 0         ADR: 3
5.  NAME: ret       KIND: var       LEVEL: 0         ADR: 4
6.  NAME: isprime   KIND: procedure LEVEL: 0         ADR: 2
7.  NAME: primes    KIND: procedure LEVEL: 0         ADR: 30
8.  TX1->
9.  NAME: i         KIND: var       LEVEL: 1         ADR: 3
10. TX2->
11.
12. Process finished with exit code 0
```

7.3.5 PL0_code4.in 语法分析结果

代码 7-17 PL0_code4.in 语法分析结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code4.in
TX0->
NAME: x          KIND: var       LEVEL: 0          ADR: 3
NAME: y          KIND: var       LEVEL: 0          ADR: 4
NAME: z          KIND: var       LEVEL: 0          ADR: 5
NAME: q          KIND: var       LEVEL: 0          ADR: 6
NAME: r          KIND: var       LEVEL: 0          ADR: 7
NAME: n          KIND: var       LEVEL: 0          ADR: 8
NAME: f          KIND: var       LEVEL: 0          ADR: 9
NAME: multiply   KIND: procedure LEVEL: 0          ADR: 2
NAME: divide     KIND: procedure LEVEL: 0          ADR: 13
NAME: gcd        KIND: procedure LEVEL: 0          ADR: 56
NAME: fact       KIND: procedure LEVEL: 0          ADR: 86
TX1->
NAME: a          KIND: var       LEVEL: 1          ADR: 3
NAME: b          KIND: var       LEVEL: 1          ADR: 4
TX2->
NAME: w          KIND: var       LEVEL: 1          ADR: 3
TX3->
NAME: g          KIND: var       LEVEL: 1          ADR: 3
TX4->

Process finished with exit code 0
```

7.3.6 PL0_code5.in 语法分析结果

代码 7-18 PL0_code5.in 语法分析结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code5.in
TX0->
NAME: c1         KIND: const     VAL: 2
NAME: v1         KIND: var       LEVEL: 0          ADR: 3
NAME: v2         KIND: var       LEVEL: 0          ADR: 4
NAME: v3         KIND: var       LEVEL: 0          ADR: 5
NAME: v4         KIND: var       LEVEL: 0          ADR: 6
NAME: p1         KIND: procedure LEVEL: 0          ADR: 2
NAME: p2         KIND: procedure LEVEL: 0          ADR: 51
TX1->
NAME: v5         KIND: var       LEVEL: 1          ADR: 3
TX2->
NAME: c2         KIND: const     VAL: 2
NAME: p3         KIND: procedure LEVEL: 1          ADR: 36
TX3->

Process finished with exit code 0
```

7.4 语法树生成结果

7.4.1 PL0_code0.in 语法树生成结果

代码 7-19 PL0_code0.in 语法树生成结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code0.in
[program]
|--[subProbgram]
```

```
|   |--[constDeclare]
|   |   |--const
|   |   |--[constDefine]
|   |   |   |--a
|   |   |   |--=
|   |   |   |__10
|   |   |__;
|   |--[varDeclare]
|   |   |--var
|   |   |--d
|   |   |--,
|   |   |--e
|   |   |--,
|   |   |--f
|   |   |__;
|   |--[procDeclare]
|   |   |--procedure
|   |   |--p
|   |   |--;
|   |   |--[subProbgram]
|   |   |   |--[varDeclare]
|   |   |   |   |--var
|   |   |   |   |--g
|   |   |   |   |__;
|   |   |   |__[beginStatement]
|   |   |      |--begin
|   |   |      |--[assignStatement]
|   |   |      |   |--d
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |--[factor]
|   |   |      |         |   |__a
|   |   |      |         |--*
|   |   |      |         |__[factor]
|   |   |      |            |__2
|   |   |      |--;
|   |   |      |--[assignStatement]
|   |   |      |   |--e
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |--[factor]
|   |   |      |         |   |__a
|   |   |      |         |--/
|   |   |      |         |__[factor]
|   |   |      |            |__3
|   |   |      |--;
|   |   |      |--[ifStatement]
|   |   |      |   |--if
|   |   |      |   |--[condition]
|   |   |      |   |   |--[expression]
|   |   |      |   |   |   |__[term]
|   |   |      |   |   |      |__[factor]
|   |   |      |   |   |         |__d
|   |   |      |   |   |--<=
|   |   |      |   |   |__[expression]
|   |   |      |   |      |__[term]
|   |   |      |   |         |__[factor]
|   |   |      |   |            |__e
|   |   |      |   |--then
|   |   |      |   |__[assignStatement]
|   |   |      |      |--f
|   |   |      |      |--:=
|   |   |      |      |__[expression]
|   |   |      |         |--[term]
```

```
|   |   |      |         |   |__[factor]
|   |   |      |         |      |__d
|   |   |      |         |--+
|   |   |      |         |__[term]
|   |   |      |            |__[factor]
|   |   |      |               |__e
|   |   |      |__end
|   |   |__;
|   |__[beginStatement]
|      |--begin
|      |--[readStatement]
|      |   |--read
|      |   |--(
|      |   |--e
|      |   |--,
|      |   |--f
|      |   |__)
|      |--;
|      |--[writeStatement]
|      |   |--write
|      |   |--(
|      |   |--[expression]
|      |   |   |__[term]
|      |   |      |__[factor]
|      |   |         |__e
|      |   |--,
|      |   |--[expression]
|      |   |   |__[term]
|      |   |      |__[factor]
|      |   |         |__f
|      |   |--,
|      |   |--[expression]
|      |   |   |__[term]
|      |   |      |__[factor]
|      |   |         |__d
|      |   |__)
|      |--;
|      |--[callStatement]
|      |   |--call
|      |   |__p
|      |--;
|      |--[whileStatement]
|      |   |--while
|      |   |--[condition]
|      |   |   |--odd
|      |   |   |__[expression]
|      |   |      |__[term]
|      |   |         |__[factor]
|      |   |            |__d
|      |   |--do
|      |   |__[assignStatement]
|      |      |--e
|      |      |--:=
|      |      |__[expression]
|      |         |---
|      |         |--[term]
|      |         |   |__[factor]
|      |         |      |__e
|      |         |--+
|      |         |__[term]
|      |            |__[factor]
|      |               |__1
|      |__end
|__.

Process finished with exit code 0
```

7.4.2 PL0_code1.in 语法树生成结果

代码 7-20 PL0_code1.in 语法树生成结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code1.in
[program]
|--[subProbgram]
|   |--[constDeclare]
|   |   |--const
|   |   |--[constDefine]
|   |   |   |--a
|   |   |   |--=
|   |   |   |__10
|   |   |__;
|   |--[varDeclare]
|   |   |--var
|   |   |--b
|   |   |--,
|   |   |--c
|   |   |__;
|   |--[procDeclare]
|   |   |--procedure
|   |   |--p
|   |   |--;
|   |   |--[subProbgram]
|   |   |   |__[beginStatement]
|   |   |      |--begin
|   |   |      |--[assignStatement]
|   |   |      |   |--c
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |--[term]
|   |   |      |      |   |__[factor]
|   |   |      |      |      |__b
|   |   |      |      |--+
|   |   |      |      |__[term]
|   |   |      |         |__[factor]
|   |   |      |            |__a
|   |   |      |__end
|   |   |__;
|   |__[beginStatement]
|      |--begin
|      |--[readStatement]
|      |   |--read
|      |   |--(
|      |   |--b
|      |   |__)
|      |--;
|      |--[whileStatement]
|      |   |--while
|      |   |--[condition]
|      |   |   |--[expression]
|      |   |   |   |__[term]
|      |   |   |      |__[factor]
|      |   |   |         |__b
|      |   |   |--#
|      |   |   |__[expression]
|      |   |      |__[term]
|      |   |         |__[factor]
|      |   |            |__0
```

```
|      |   |--do
|      |   |__[beginStatement]
|      |      |--begin
|      |      |--[callStatement]
|      |      |   |--call
|      |      |   |__p
|      |      |--;
|      |      |--[writeStatement]
|      |      |   |--write
|      |      |   |--(
|      |      |   |--[expression]
|      |      |   |   |__[term]
|      |      |   |      |--[factor]
|      |      |   |      |   |__2
|      |      |   |      |--*
|      |      |   |      |__[factor]
|      |      |   |         |__c
|      |      |   |__)
|      |      |--;
|      |      |--[readStatement]
|      |      |   |--read
|      |      |   |--(
|      |      |   |--b
|      |      |   |__)
|      |      |--;
|      |      |__end
|      |__end
|__.

Process finished with exit code 0
```

7.4.3 PL0_code2.in 语法树生成结果

代码 7-21 PL0_code2.in 语法树生成结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users
\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code2.in
[program]
|--[subProbgram]
|   |--[varDeclare]
|   |   |--var
|   |   |--x
|   |   |--,
|   |   |--squ
|   |   |__;
|   |--[procDeclare]
|   |   |--procedure
|   |   |--square
|   |   |--;
|   |   |--[subProbgram]
|   |   |   |__[beginStatement]
|   |   |      |--begin
|   |   |      |--[assignStatement]
|   |   |      |   |--squ
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |--[factor]
|   |   |      |         |   |__x
|   |   |      |         |--*
|   |   |      |         |__[factor]
|   |   |      |            |__x
|   |   |      |__end
```

```
|   |   |__;
|   |__[beginStatement]
|      |--begin
|      |--[assignStatement]
|      |   |--x
|      |   |--:=
|      |   |__[expression]
|      |      |__[term]
|      |         |__[factor]
|      |            |__1
|      |--;
|      |--[whileStatement]
|      |   |--while
|      |   |--[condition]
|      |   |   |--[expression]
|      |   |   |   |__[term]
|      |   |   |      |__[factor]
|      |   |   |         |__x
|      |   |   |--<=
|      |   |   |__[expression]
|      |   |      |__[term]
|      |   |         |__[factor]
|      |   |            |__10
|      |   |--do
|      |   |__[beginStatement]
|      |      |--begin
|      |      |--[callStatement]
|      |      |   |--call
|      |      |   |__square
|      |      |--;
|      |      |--[writeStatement]
|      |      |   |--write
|      |      |   |--(
|      |      |   |--[expression]
|      |      |   |   |__[term]
|      |      |   |      |__[factor]
|      |      |   |         |__squ
|      |      |   |__)
|      |      |--;
|      |      |--[assignStatement]
|      |      |   |--x
|      |      |   |--:=
|      |      |   |__[expression]
|      |      |      |--[term]
|      |      |      |   |__[factor]
|      |      |      |      |__x
|      |      |      |--+
|      |      |      |__[term]
|      |      |         |__[factor]
|      |      |            |__1
|      |      |__end
|      |__end
|__.

Process finished with exit code 0
```

7.4.4 PL0_code3.in 语法树生成结果

代码 7-22 PL0_code3.in 语法树生成结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code3.in
[program]
|--[subProbgram]
|   |--[constDeclare]
|   |   |--const
|   |   |--[constDefine]
|   |   |   |--max
|   |   |   |--=
|   |   |   |__100
|   |   |__;
|   |--[varDeclare]
|   |   |--var
|   |   |--arg
|   |   |--,
|   |   |--ret
|   |   |__;
|   |--[procDeclare]
|   |   |--procedure
|   |   |--isprime
|   |   |--;
|   |   |--[subProbgram]
|   |   |   |--[varDeclare]
|   |   |   |   |--var
|   |   |   |   |--i
|   |   |   |   |__;
|   |   |   |__[beginStatement]
|   |   |      |--begin
|   |   |      |--[assignStatement]
|   |   |      |   |--ret
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |__[factor]
|   |   |      |            |__1
|   |   |      |--;
|   |   |      |--[assignStatement]
|   |   |      |   |--i
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |__[factor]
|   |   |      |            |__2
|   |   |      |--;
|   |   |      |--[whileStatement]
|   |   |      |   |--while
|   |   |      |   |--[condition]
|   |   |      |   |   |--[expression]
|   |   |      |   |   |   |__[term]
|   |   |      |   |   |      |__[factor]
|   |   |      |   |   |         |__i
|   |   |      |   |   |--<
|   |   |      |   |   |__[expression]
|   |   |      |   |      |__[term]
|   |   |      |   |         |__[factor]
|   |   |      |   |            |__arg
|   |   |      |   |--do
|   |   |      |   |__[beginStatement]
|   |   |      |      |--begin
|   |   |      |      |--[ifStatement]
|   |   |      |      |   |--if
|   |   |      |      |   |--[condition]
|   |   |      |      |   |   |--[expression]
|   |   |      |      |   |   |   |__[term]
|   |   |      |      |   |   |      |--[factor]
|   |   |      |      |   |   |      |   |__arg
```

```
|   |   |      |      |   |   |      |--/
|   |   |      |      |   |   |      |--[factor]
|   |   |      |      |   |   |      |   |__i
|   |   |      |      |   |   |      |--*
|   |   |      |      |   |   |      |__[factor]
|   |   |      |      |   |   |         |__i
|   |   |      |      |   |   |--=
|   |   |      |      |   |   |__[expression]
|   |   |      |      |   |      |__[term]
|   |   |      |      |   |         |__[factor]
|   |   |      |      |   |            |__arg
|   |   |      |      |   |--then
|   |   |      |      |   |__[beginStatement]
|   |   |      |      |      |--begin
|   |   |      |      |      |--[assignStatement]
|   |   |      |      |      |   |--ret
|   |   |      |      |      |   |--:=
|   |   |      |      |      |   |__[expression]
|   |   |      |      |      |      |__[term]
|   |   |      |      |      |         |__[factor]
|   |   |      |      |      |            |__0
|   |   |      |      |      |--;
|   |   |      |      |      |--[assignStatement]
|   |   |      |      |      |   |--i
|   |   |      |      |      |   |--:=
|   |   |      |      |      |   |__[expression]
|   |   |      |      |      |      |__[term]
|   |   |      |      |      |         |__[factor]
|   |   |      |      |      |            |__arg
|   |   |      |      |      |__end
|   |   |      |      |--;
|   |   |      |      |--[assignStatement]
|   |   |      |      |   |--i
|   |   |      |      |   |--:=
|   |   |      |      |   |__[expression]
|   |   |      |      |      |--[term]
|   |   |      |      |      |   |__[factor]
|   |   |      |      |      |      |__i
|   |   |      |      |      |--+
|   |   |      |      |      |__[term]
|   |   |      |      |         |__[factor]
|   |   |      |      |            |__1
|   |   |      |      |__end
|   |   |      |__end
|   |   |--;
|   |   |--procedure
|   |   |--primes
|   |   |--;
|   |   |--[subProbgram]
|   |   |   |__[beginStatement]
|   |   |      |--begin
|   |   |      |--[assignStatement]
|   |   |      |   |--arg
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |__[factor]
|   |   |      |            |__2
|   |   |      |--;
|   |   |      |--[whileStatement]
|   |   |      |   |--while
|   |   |      |   |--[condition]
|   |   |      |   |   |--[expression]
|   |   |      |   |   |   |__[term]
|   |   |      |   |   |      |__[factor]
|   |   |      |   |   |         |__arg
```

```
|   |   |      |   |   |--<
|   |   |      |   |   |__[expression]
|   |   |      |   |      |__[term]
|   |   |      |   |         |__[factor]
|   |   |      |   |            |__max
|   |   |      |   |--do
|   |   |      |   |__[beginStatement]
|   |   |      |      |--begin
|   |   |      |      |--[callStatement]
|   |   |      |      |   |--call
|   |   |      |      |   |__isprime
|   |   |      |      |--;
|   |   |      |      |--[ifStatement]
|   |   |      |      |   |--if
|   |   |      |      |   |--[condition]
|   |   |      |      |   |   |--[expression]
|   |   |      |      |   |   |   |__[term]
|   |   |      |      |   |   |      |__[factor]
|   |   |      |      |   |   |         |__ret
|   |   |      |      |   |   |--=
|   |   |      |      |   |   |__[expression]
|   |   |      |      |   |      |__[term]
|   |   |      |      |   |         |__[factor]
|   |   |      |      |   |            |__1
|   |   |      |      |   |--then
|   |   |      |      |   |__[writeStatement]
|   |   |      |      |      |--write
|   |   |      |      |      |--(
|   |   |      |      |      |--[expression]
|   |   |      |      |      |   |__[term]
|   |   |      |      |      |      |__[factor]
|   |   |      |      |      |         |__arg
|   |   |      |      |      |__)
|   |   |      |      |--;
|   |   |      |      |--[assignStatement]
|   |   |      |      |   |--arg
|   |   |      |      |   |--:=
|   |   |      |      |   |__[expression]
|   |   |      |      |      |--[term]
|   |   |      |      |      |   |__[factor]
|   |   |      |      |      |      |__arg
|   |   |      |      |      |--+
|   |   |      |      |      |__[term]
|   |   |      |      |         |__[factor]
|   |   |      |      |            |__1
|   |   |      |      |__end
|   |   |      |__end
|   |   |__;
|   |__[callStatement]
|      |--call
|      |__primes
|__.

Process finished with exit code 0
```

7.4.5 PL0_code4.in 语法树生成结果

代码 7-23 PL0_code4.in 语法树生成结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users
\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code4.in
```

```
[program]
|--[subProbgram]
|   |--[varDeclare]
|   |   |--var
|   |   |--x
|   |   |--,
|   |   |--y
|   |   |--,
|   |   |--z
|   |   |--,
|   |   |--q
|   |   |--,
|   |   |--r
|   |   |--,
|   |   |--n
|   |   |--,
|   |   |--f
|   |   |__;
|   |--[procDeclare]
|   |   |--procedure
|   |   |--multiply
|   |   |--;
|   |   |--[subProbgram]
|   |   |   |--[varDeclare]
|   |   |   |   |--var
|   |   |   |   |--a
|   |   |   |   |--,
|   |   |   |   |--b
|   |   |   |   |__;
|   |   |   |__[beginStatement]
|   |   |      |--begin
|   |   |      |--[assignStatement]
|   |   |      |   |--a
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |__[factor]
|   |   |      |            |__x
|   |   |      |--;
|   |   |      |--[assignStatement]
|   |   |      |   |--b
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |__[factor]
|   |   |      |            |__y
|   |   |      |--;
|   |   |      |--[assignStatement]
|   |   |      |   |--z
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |--[factor]
|   |   |      |         |   |__x
|   |   |      |         |--*
|   |   |      |         |__[factor]
|   |   |      |            |__y
|   |   |      |__end
|   |   |--;
|   |   |--procedure
|   |   |--divide
|   |   |--;
|   |   |--[subProbgram]
|   |   |   |--[varDeclare]
|   |   |   |   |--var
|   |   |   |   |--w
```

```
|   |   |   |   |__;
|   |   |   |__[beginStatement]
|   |   |      |--begin
|   |   |      |--[assignStatement]
|   |   |      |   |--r
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |__[factor]
|   |   |      |            |__x
|   |   |      |--;
|   |   |      |--[assignStatement]
|   |   |      |   |--q
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |__[factor]
|   |   |      |            |__0
|   |   |      |--;
|   |   |      |--[assignStatement]
|   |   |      |   |--w
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |__[factor]
|   |   |      |            |__y
|   |   |      |--;
|   |   |      |--[whileStatement]
|   |   |      |   |--while
|   |   |      |   |--[condition]
|   |   |      |   |   |--[expression]
|   |   |      |   |   |   |__[term]
|   |   |      |   |   |      |__[factor]
|   |   |      |   |   |         |__w
|   |   |      |   |   |--<=
|   |   |      |   |   |__[expression]
|   |   |      |   |      |__[term]
|   |   |      |   |         |__[factor]
|   |   |      |   |            |__r
|   |   |      |   |--do
|   |   |      |   |__[assignStatement]
|   |   |      |      |--w
|   |   |      |      |--:=
|   |   |      |      |__[expression]
|   |   |      |         |__[term]
|   |   |      |            |--[factor]
|   |   |      |            |   |__2
|   |   |      |            |--*
|   |   |      |            |__[factor]
|   |   |      |               |__w
|   |   |      |--;
|   |   |      |--[whileStatement]
|   |   |      |   |--while
|   |   |      |   |--[condition]
|   |   |      |   |   |--[expression]
|   |   |      |   |   |   |__[term]
|   |   |      |   |   |      |__[factor]
|   |   |      |   |   |         |__w
|   |   |      |   |   |-->
|   |   |      |   |   |__[expression]
|   |   |      |   |      |__[term]
|   |   |      |   |         |__[factor]
|   |   |      |   |            |__y
|   |   |      |   |--do
|   |   |      |   |__[beginStatement]
|   |   |      |      |--begin
```

```
|   |   |      |      |--[assignStatement]
|   |   |      |      |   |--q
|   |   |      |      |   |--:=
|   |   |      |      |   |__[expression]
|   |   |      |      |      |__[term]
|   |   |      |      |         |--[factor]
|   |   |      |      |         |   |__2
|   |   |      |      |         |--*
|   |   |      |      |         |__[factor]
|   |   |      |      |            |__q
|   |   |      |      |--;
|   |   |      |      |--[assignStatement]
|   |   |      |      |   |--w
|   |   |      |      |   |--:=
|   |   |      |      |   |__[expression]
|   |   |      |      |      |__[term]
|   |   |      |      |         |--[factor]
|   |   |      |      |         |   |__w
|   |   |      |      |         |--/
|   |   |      |      |         |__[factor]
|   |   |      |      |            |__2
|   |   |      |      |--;
|   |   |      |      |--[ifStatement]
|   |   |      |      |   |--if
|   |   |      |      |   |--[condition]
|   |   |      |      |   |   |--[expression]
|   |   |      |      |   |   |   |__[term]
|   |   |      |      |   |   |      |__[factor]
|   |   |      |      |   |   |         |__w
|   |   |      |      |   |   |--<=
|   |   |      |      |   |   |__[expression]
|   |   |      |      |   |      |__[term]
|   |   |      |      |   |         |__[factor]
|   |   |      |      |   |            |__r
|   |   |      |      |   |--then
|   |   |      |      |   |__[beginStatement]
|   |   |      |      |      |--begin
|   |   |      |      |      |--[assignStatement]
|   |   |      |      |      |   |--r
|   |   |      |      |      |   |--:=
|   |   |      |      |      |   |__[expression]
|   |   |      |      |      |      |--[term]
|   |   |      |      |      |      |   |__[factor]
|   |   |      |      |      |      |      |__r
|   |   |      |      |      |      |---
|   |   |      |      |      |      |__[term]
|   |   |      |      |      |         |__[factor]
|   |   |      |      |      |            |__w
|   |   |      |      |      |--;
|   |   |      |      |      |--[assignStatement]
|   |   |      |      |      |   |--q
|   |   |      |      |      |   |--:=
|   |   |      |      |      |   |__[expression]
|   |   |      |      |      |      |--[term]
|   |   |      |      |      |      |   |__[factor]
|   |   |      |      |      |      |      |__q
|   |   |      |      |      |      |--+
|   |   |      |      |      |      |__[term]
|   |   |      |      |      |         |__[factor]
|   |   |      |      |      |            |__1
|   |   |      |      |      |__end
|   |   |      |      |__end
|   |   |      |__end
|   |   |--;
|   |   |--procedure
|   |   |--gcd
```

```
|   |   |--;
|   |   |--[subProbgram]
|   |   |   |--[varDeclare]
|   |   |   |   |--var
|   |   |   |   |--g
|   |   |   |   |__;
|   |   |   |__[beginStatement]
|   |   |      |--begin
|   |   |      |--[assignStatement]
|   |   |      |   |--f
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |__[factor]
|   |   |      |            |__x
|   |   |      |--;
|   |   |      |--[assignStatement]
|   |   |      |   |--g
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |__[factor]
|   |   |      |            |__y
|   |   |      |--;
|   |   |      |--[whileStatement]
|   |   |      |   |--while
|   |   |      |   |--[condition]
|   |   |      |   |   |--[expression]
|   |   |      |   |   |   |__[term]
|   |   |      |   |   |      |__[factor]
|   |   |      |   |   |         |__f
|   |   |      |   |   |--#
|   |   |      |   |   |__[expression]
|   |   |      |   |      |__[term]
|   |   |      |   |         |__[factor]
|   |   |      |   |            |__g
|   |   |      |   |--do
|   |   |      |   |__[beginStatement]
|   |   |      |      |--begin
|   |   |      |      |--[ifStatement]
|   |   |      |      |   |--if
|   |   |      |      |   |--[condition]
|   |   |      |      |   |   |--[expression]
|   |   |      |      |   |   |   |__[term]
|   |   |      |      |   |   |      |__[factor]
|   |   |      |      |   |   |         |__f
|   |   |      |      |   |   |--<
|   |   |      |      |   |   |__[expression]
|   |   |      |      |   |      |__[term]
|   |   |      |      |   |         |__[factor]
|   |   |      |      |   |            |__g
|   |   |      |      |   |--then
|   |   |      |      |   |__[assignStatement]
|   |   |      |      |      |--g
|   |   |      |      |      |--:=
|   |   |      |      |      |__[expression]
|   |   |      |      |         |--[term]
|   |   |      |      |         |   |__[factor]
|   |   |      |      |         |      |__g
|   |   |      |      |         |---
|   |   |      |      |         |__[term]
|   |   |      |      |            |__[factor]
|   |   |      |      |               |__f
|   |   |      |      |--;
|   |   |      |      |--[ifStatement]
|   |   |      |      |   |--if
```

```
|   |   |      |      |   |--[condition]
|   |   |      |      |   |   |--[expression]
|   |   |      |      |   |   |   |__[term]
|   |   |      |      |   |   |      |__[factor]
|   |   |      |      |   |   |         |__g
|   |   |      |      |   |   |--<
|   |   |      |      |   |   |__[expression]
|   |   |      |      |   |      |__[term]
|   |   |      |      |   |         |__[factor]
|   |   |      |      |   |            |__f
|   |   |      |      |   |--then
|   |   |      |      |   |__[assignStatement]
|   |   |      |      |      |--f
|   |   |      |      |      |--:=
|   |   |      |      |      |__[expression]
|   |   |      |      |         |--[term]
|   |   |      |      |         |   |__[factor]
|   |   |      |      |         |      |__f
|   |   |      |      |         |---
|   |   |      |      |         |__[term]
|   |   |      |      |            |__[factor]
|   |   |      |      |               |__g
|   |   |      |      |__end
|   |   |      |--;
|   |   |      |--[assignStatement]
|   |   |      |   |--z
|   |   |      |   |--:=
|   |   |      |   |__[expression]
|   |   |      |      |__[term]
|   |   |      |         |__[factor]
|   |   |      |            |__f
|   |   |      |__end
|   |   |--;
|   |   |--procedure
|   |   |--fact
|   |   |--;
|   |   |--[subProbgram]
|   |   |   |__[beginStatement]
|   |   |      |--begin
|   |   |      |--[ifStatement]
|   |   |      |   |--if
|   |   |      |   |--[condition]
|   |   |      |   |   |--[expression]
|   |   |      |   |   |   |__[term]
|   |   |      |   |   |      |__[factor]
|   |   |      |   |   |         |__n
|   |   |      |   |   |-->
|   |   |      |   |   |__[expression]
|   |   |      |   |      |__[term]
|   |   |      |   |         |__[factor]
|   |   |      |   |            |__1
|   |   |      |   |--then
|   |   |      |   |__[beginStatement]
|   |   |      |      |--begin
|   |   |      |      |--[assignStatement]
|   |   |      |      |   |--f
|   |   |      |      |   |--:=
|   |   |      |      |   |__[expression]
|   |   |      |      |      |__[term]
|   |   |      |      |         |--[factor]
|   |   |      |      |         |   |__n
|   |   |      |      |         |--*
|   |   |      |      |         |__[factor]
|   |   |      |      |            |__f
|   |   |      |      |--;
|   |   |      |      |--[assignStatement]
```

```
|   |   |      |      |   |--n
|   |   |      |      |   |--:=
|   |   |      |      |   |__[expression]
|   |   |      |      |      |--[term]
|   |   |      |      |      |   |__[factor]
|   |   |      |      |      |      |__n
|   |   |      |      |      |---
|   |   |      |      |      |__[term]
|   |   |      |      |         |__[factor]
|   |   |      |      |            |__1
|   |   |      |      |--;
|   |   |      |      |--[callStatement]
|   |   |      |      |   |--call
|   |   |      |      |   |__fact
|   |   |      |      |__end
|   |   |      |__end
|   |   |__;
|   |__[beginStatement]
|      |--begin
|      |--[readStatement]
|      |   |--read
|      |   |--(
|      |   |--x
|      |   |__)
|      |--;
|      |--[readStatement]
|      |   |--read
|      |   |--(
|      |   |--y
|      |   |__)
|      |--;
|      |--[callStatement]
|      |   |--call
|      |   |__multiply
|      |--;
|      |--[writeStatement]
|      |   |--write
|      |   |--(
|      |   |--[expression]
|      |   |   |__[term]
|      |   |      |__[factor]
|      |   |         |__z
|      |   |__)
|      |--;
|      |--[readStatement]
|      |   |--read
|      |   |--(
|      |   |--x
|      |   |__)
|      |--;
|      |--[readStatement]
|      |   |--read
|      |   |--(
|      |   |--y
|      |   |__)
|      |--;
|      |--[callStatement]
|      |   |--call
|      |   |__divide
|      |--;
|      |--[writeStatement]
|      |   |--write
|      |   |--(
|      |   |--[expression]
|      |   |   |__[term]
|      |   |      |__[factor]
```

```
|      |   |         |__q
|      |   |__)
|      |--;
|      |--[writeStatement]
|      |   |--write
|      |   |--(
|      |   |--[expression]
|      |   |   |__[term]
|      |   |      |__[factor]
|      |   |         |__r
|      |   |__)
|      |--;
|      |--[readStatement]
|      |   |--read
|      |   |--(
|      |   |--x
|      |   |__)
|      |--;
|      |--[readStatement]
|      |   |--read
|      |   |--(
|      |   |--y
|      |   |__)
|      |--;
|      |--[callStatement]
|      |   |--call
|      |   |__gcd
|      |--;
|      |--[writeStatement]
|      |   |--write
|      |   |--(
|      |   |--[expression]
|      |   |   |__[term]
|      |   |      |__[factor]
|      |   |         |__z
|      |   |__)
|      |--;
|      |--[readStatement]
|      |   |--read
|      |   |--(
|      |   |--n
|      |   |__)
|      |--;
|      |--[assignStatement]
|      |   |--f
|      |   |--:=
|      |   |__[expression]
|      |      |__[term]
|      |         |__[factor]
|      |            |__1
|      |--;
|      |--[callStatement]
|      |   |--call
|      |   |__fact
|      |--;
|      |--[writeStatement]
|      |   |--write
|      |   |--(
|      |   |--[expression]
|      |   |   |__[term]
|      |   |      |__[factor]
|      |   |         |__f
|      |   |__)
|      |__end
|__.
```

```
464. Process finished with exit code 0
```

7.4.6 PL0_code5.in 语法树生成结果

代码 7-24 PL0_code5.in 语法树生成结果

```
1.  C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users
    \TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code5.in
2.  [program]
3.  |--[subProbgram]
4.  |   |--[constDeclare]
5.  |   |   |--const
6.  |   |   |--[constDefine]
7.  |   |   |   |--c1
8.  |   |   |   |--=
9.  |   |   |   |__2
10. |   |   |__;
11. |   |--[varDeclare]
12. |   |   |--var
13. |   |   |--v1
14. |   |   |--,
15. |   |   |--v2
16. |   |   |--,
17. |   |   |--v3
18. |   |   |--,
19. |   |   |--v4
20. |   |   |__;
21. |   |--[procDeclare]
22. |   |   |--procedure
23. |   |   |--p1
24. |   |   |--;
25. |   |   |--[subProbgram]
26. |   |   |   |--[varDeclare]
27. |   |   |   |   |--var
28. |   |   |   |   |--v5
29. |   |   |   |   |__;
30. |   |   |   |__[beginStatement]
31. |   |   |      |--begin
32. |   |   |      |--[assignStatement]
33. |   |   |      |   |--v5
34. |   |   |      |   |--:=
35. |   |   |      |   |__[expression]
36. |   |   |      |      |__[term]
37. |   |   |      |         |__[factor]
38. |   |   |      |            |__2
39. |   |   |      |--;
40. |   |   |      |--[writeStatement]
41. |   |   |      |   |--write
42. |   |   |      |   |--(
43. |   |   |      |   |--[expression]
44. |   |   |      |   |   |--[term]
45. |   |   |      |   |   |   |--[factor]
46. |   |   |      |   |   |   |   |__v5
47. |   |   |      |   |   |   |--/
48. |   |   |      |   |   |   |__[factor]
49. |   |   |      |   |   |      |__2
50. |   |   |      |   |   |--+
51. |   |   |      |   |   |--[term]
52. |   |   |      |   |   |   |__[factor]
53. |   |   |      |   |   |      |__2
54. |   |   |      |   |   |---
```

```
|   |   |      |   |   |__[term]
|   |   |      |   |      |__[factor]
|   |   |      |   |         |__1
|   |   |      |   |__)
|   |   |      |--;
|   |   |      |--[whileStatement]
|   |   |      |   |--while
|   |   |      |   |--[condition]
|   |   |      |   |   |--[expression]
|   |   |      |   |   |   |__[term]
|   |   |      |   |   |      |__[factor]
|   |   |      |   |   |         |__v3
|   |   |      |   |   |--#
|   |   |      |   |   |__[expression]
|   |   |      |   |      |__[term]
|   |   |      |   |         |__[factor]
|   |   |      |   |            |__0
|   |   |      |   |--do
|   |   |      |   |__[beginStatement]
|   |   |      |      |--begin
|   |   |      |      |--[assignStatement]
|   |   |      |      |   |--v4
|   |   |      |      |   |--:=
|   |   |      |      |   |__[expression]
|   |   |      |      |      |__[term]
|   |   |      |      |         |--[factor]
|   |   |      |      |         |   |__v1
|   |   |      |      |         |--/
|   |   |      |      |         |__[factor]
|   |   |      |      |            |__v2
|   |   |      |      |--;
|   |   |      |      |--[assignStatement]
|   |   |      |      |   |--v3
|   |   |      |      |   |--:=
|   |   |      |      |   |__[expression]
|   |   |      |      |      |--[term]
|   |   |      |      |      |   |__[factor]
|   |   |      |      |      |      |__v1
|   |   |      |      |      |---
|   |   |      |      |      |__[term]
|   |   |      |      |         |--[factor]
|   |   |      |      |         |   |__v4
|   |   |      |      |         |--*
|   |   |      |      |         |__[factor]
|   |   |      |      |            |__v2
|   |   |      |      |--;
|   |   |      |      |--[assignStatement]
|   |   |      |      |   |--v1
|   |   |      |      |   |--:=
|   |   |      |      |   |__[expression]
|   |   |      |      |      |__[term]
|   |   |      |      |         |__[factor]
|   |   |      |      |            |__v2
|   |   |      |      |--;
|   |   |      |      |--[assignStatement]
|   |   |      |      |   |--v2
|   |   |      |      |   |--:=
|   |   |      |      |   |__[expression]
|   |   |      |      |      |__[term]
|   |   |      |      |         |__[factor]
|   |   |      |      |            |__v3
|   |   |      |      |--;
|   |   |      |      |__end
|   |   |      |--;
|   |   |      |__end
|   |   |--;
```

```
|   |   |--procedure
|   |   |--p2
|   |   |--;
|   |   |--[subProbgram]
|   |   |   |--[constDeclare]
|   |   |   |   |--const
|   |   |   |   |--[constDefine]
|   |   |   |   |   |--c2
|   |   |   |   |   |--:=
|   |   |   |   |   |__2
|   |   |   |   |__;
|   |   |   |--[procDeclare]
|   |   |   |   |--procedure
|   |   |   |   |--p3
|   |   |   |   |--;
|   |   |   |   |--[subProbgram]
|   |   |   |   |   |__[beginStatement]
|   |   |   |   |      |--begin
|   |   |   |   |      |--[ifStatement]
|   |   |   |   |      |   |--if
|   |   |   |   |      |   |--[condition]
|   |   |   |   |      |   |   |--[expression]
|   |   |   |   |      |   |   |   |__[term]
|   |   |   |   |      |   |   |      |__[factor]
|   |   |   |   |      |   |   |         |__v1
|   |   |   |   |      |   |   |--#
|   |   |   |   |      |   |   |__[expression]
|   |   |   |   |      |   |      |__[term]
|   |   |   |   |      |   |         |__[factor]
|   |   |   |   |      |   |            |__1
|   |   |   |   |      |   |--then
|   |   |   |   |      |   |__[beginStatement]
|   |   |   |   |      |      |--begin
|   |   |   |   |      |      |--[assignStatement]
|   |   |   |   |      |      |   |--v1
|   |   |   |   |      |      |   |--:=
|   |   |   |   |      |      |   |__[expression]
|   |   |   |   |      |      |      |--[term]
|   |   |   |   |      |      |      |   |__[factor]
|   |   |   |   |      |      |      |      |__v1
|   |   |   |   |      |      |      |---
|   |   |   |   |      |      |      |__[term]
|   |   |   |   |      |      |         |__[factor]
|   |   |   |   |      |      |            |__1
|   |   |   |   |      |      |--;
|   |   |   |   |      |      |--[assignStatement]
|   |   |   |   |      |      |   |--v2
|   |   |   |   |      |      |   |--:=
|   |   |   |   |      |      |   |__[expression]
|   |   |   |   |      |      |      |__[term]
|   |   |   |   |      |      |         |--[factor]
|   |   |   |   |      |      |         |   |__v2
|   |   |   |   |      |      |         |--*
|   |   |   |   |      |      |         |__[factor]
|   |   |   |   |      |      |            |__v1
|   |   |   |   |      |      |--;
|   |   |   |   |      |      |--[callStatement]
|   |   |   |   |      |      |   |--call
|   |   |   |   |      |      |   |__p3
|   |   |   |   |      |      |--;
|   |   |   |   |      |      |__end
|   |   |   |   |      |--;
|   |   |   |   |      |__end
|   |   |   |   |__;
|   |   |   |__[beginStatement]
|   |   |      |--begin
```

```
|   |   |      |--[callStatement]
|   |   |      |   |--call
|   |   |      |   |__p3
|   |   |      |--;
|   |   |      |--[ifStatement]
|   |   |      |   |--if
|   |   |      |   |--[condition]
|   |   |      |   |   |--odd
|   |   |      |   |   |__[expression]
|   |   |      |   |      |__[term]
|   |   |      |   |         |__[factor]
|   |   |      |   |            |__c2
|   |   |      |   |--then
|   |   |      |   |__[writeStatement]
|   |   |      |      |--write
|   |   |      |      |--(
|   |   |      |      |--[expression]
|   |   |      |      |   |__[term]
|   |   |      |      |      |__[factor]
|   |   |      |      |         |__c2
|   |   |      |      |__)
|   |   |      |--;
|   |   |      |--[ifStatement]
|   |   |      |   |--if
|   |   |      |   |--[condition]
|   |   |      |   |   |--[expression]
|   |   |      |   |   |   |__[term]
|   |   |      |   |   |      |__[factor]
|   |   |      |   |   |         |__c2
|   |   |      |   |   |--=
|   |   |      |   |   |__[expression]
|   |   |      |   |      |__[term]
|   |   |      |   |         |__[factor]
|   |   |      |   |            |__2
|   |   |      |   |--then
|   |   |      |   |__[writeStatement]
|   |   |      |      |--write
|   |   |      |      |--(
|   |   |      |      |--[expression]
|   |   |      |      |   |--[term]
|   |   |      |      |   |   |__[factor]
|   |   |      |      |   |      |__c2
|   |   |      |      |   |--+
|   |   |      |      |   |__[term]
|   |   |      |      |      |__[factor]
|   |   |      |      |         |__1
|   |   |      |      |__)
|   |   |      |__end
|   |   |__;
|   |__[beginStatement]
|      |--begin
|      |--[readStatement]
|      |   |--read
|      |   |--(
|      |   |--v1
|      |   |--,
|      |   |--v2
|      |   |__)
|      |--;
|      |--[ifStatement]
|      |   |--if
|      |   |--[condition]
|      |   |   |--[expression]
|      |   |   |   |__[term]
|      |   |   |      |__[factor]
|      |   |   |         |__v1
```

```
|      |   |   |--<
|      |   |   |__[expression]
|      |   |      |__[term]
|      |   |         |__[factor]
|      |   |            |__v2
|      |   |--then
|      |   |__[beginStatement]
|      |      |--begin
|      |      |--[assignStatement]
|      |      |   |--v3
|      |      |   |--:=
|      |      |   |__[expression]
|      |      |      |__[term]
|      |      |         |__[factor]
|      |      |            |__v1
|      |      |--;
|      |      |--[assignStatement]
|      |      |   |--v1
|      |      |   |--:=
|      |      |   |__[expression]
|      |      |      |__[term]
|      |      |         |__[factor]
|      |      |            |__v2
|      |      |--;
|      |      |--[assignStatement]
|      |      |   |--v2
|      |      |   |--:=
|      |      |   |__[expression]
|      |      |      |__[term]
|      |      |         |__[factor]
|      |      |            |__v3
|      |      |--;
|      |      |__end
|      |--;
|      |--[beginStatement]
|      |   |--begin
|      |   |--;
|      |   |--[assignStatement]
|      |   |   |--v3
|      |   |   |--:=
|      |   |   |__[expression]
|      |   |      |__[term]
|      |   |         |__[factor]
|      |   |            |__1
|      |   |--;
|      |   |--[callStatement]
|      |   |   |--call
|      |   |   |__p1
|      |   |--;
|      |   |--[writeStatement]
|      |   |   |--write
|      |   |   |--(
|      |   |   |--[expression]
|      |   |   |   |__[term]
|      |   |   |      |__[factor]
|      |   |   |         |__c1
|      |   |   |--,
|      |   |   |--[expression]
|      |   |   |   |__[term]
|      |   |   |      |--[factor]
|      |   |   |      |   |__c1
|      |   |   |      |--*
|      |   |   |      |__[factor]
|      |   |   |         |__v1
|      |   |   |--,
|      |   |   |--[expression]
```

```
|       |   |   |   |__[term]
|       |   |   |       |--[factor]
|       |   |   |       |   |__c1
|       |   |   |       |--*
|       |   |   |       |--[factor]
|       |   |   |       |   |__v1
|       |   |   |       |--/
|       |   |   |       |__[factor]
|       |   |   |          |__2
|       |   |   |__)
|       |   |--;
|       |   |__end
|       |--;
|       |--[readStatement]
|       |   |--read
|       |   |--(
|       |   |--v1
|       |   |__)
|       |--;
|       |--[assignStatement]
|       |   |--v2
|       |   |--:=
|       |   |__[expression]
|       |      |__[term]
|       |         |__[factor]
|       |            |__v1
|       |--;
|       |--[callStatement]
|       |   |--call
|       |   |__p2
|       |--;
|       |--[writeStatement]
|       |   |--write
|       |   |--(
|       |   |--[expression]
|       |   |   |__[term]
|       |   |      |__[factor]
|       |   |         |__v2
|       |   |__)
|       |--;
|       |__end
|__.

Process finished with exit code 0
```

7.5 中间代码生成结果

7.5.1 PL0_code0.in 中间代码生成结果

代码 7-25 PL0_code0.in 中间代码生成结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code0.in
0       JMP     0       20
1       JMP     0       2
2       INT     0       4
3       LIT     0       10
4       LIT     0       2
5       OPR     0       4(MUL)
```

```
8.  6       STO     1       3
9.  7       LIT     0       10
10. 8       LIT     0       3
11. 9       OPR     0       5(DIV)
12. 10      STO     1       4
13. 11      LOD     1       3
14. 12      LOD     1       4
15. 13      OPR     0       6(LEQ)
16. 14      JPC     0       19
17. 15      LOD     1       3
18. 16      LOD     1       4
19. 17      OPR     0       2(ADD)
20. 18      STO     1       5
21. 19      OPR     0       0(RET)
22. 20      INT     0       7
23. 21      OPR     0       16(READ)
24. 22      STO     0       4
25. 23      OPR     0       16(READ)
26. 24      STO     0       5
27. 25      LOD     0       4
28. 26      OPR     0       14(WRITE)
29. 27      LOD     0       5
30. 28      OPR     0       14(WRITE)
31. 29      LOD     0       3
32. 30      OPR     0       14(WRITE)
33. 31      OPR     0       15(LINE)
34. 32      CAL     0       2
35. 33      LOD     0       3
36. 34      OPR     0       6(LEQ)
37. 35      JPC     0       42
38. 36      LOD     0       4
39. 37      OPR     0       1(NEG)
40. 38      LIT     0       1
41. 39      OPR     0       2(ADD)
42. 40      STO     0       4
43. 41      JMP     0       33
44. 42      OPR     0       0(RET)
45.
46. Process finished with exit code 0
```

7.5.2 PL0_code1.in 中间代码生成结果

代码 7-26 PL0_code1.in 中间代码生成结果

```
1.  C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\
    TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code1.in
2.  0       JMP     0       8
3.  1       JMP     0       2
4.  2       INT     0       3
5.  3       LOD     1       3
6.  4       LIT     0       10
7.  5       OPR     0       2(ADD)
8.  6       STO     1       4
9.  7       OPR     0       0(RET)
10. 8       INT     0       6
11. 9       OPR     0       16(READ)
12. 10      STO     0       3
13. 11      LOD     0       3
14. 12      LIT     0       0
15. 13      OPR     0       9(NEQ)
16. 14      JPC     0       24
17. 15      CAL     0       2
18. 16      LIT     0       2
```

```
19. 17      LOD     0       4
20. 18      OPR     0       4(MUL)
21. 19      OPR     0       14(WRITE)
22. 20      OPR     0       15(LINE)
23. 21      OPR     0       16(READ)
24. 22      STO     0       3
25. 23      JMP     0       11
26. 24      OPR     0       0(RET)
27.
28. Process finished with exit code 0
```

7.5.3 PL0_code2.in 中间代码生成结果

代码 7-27 PL0_code2.in 中间代码生成结果

```
1.  C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\
    TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code2.in
2.  0       JMP     0       8
3.  1       JMP     0       2
4.  2       INT     0       3
5.  3       LOD     1       3
6.  4       LOD     1       3
7.  5       OPR     0       4(MUL)
8.  6       STO     1       4
9.  7       OPR     0       0(RET)
10. 8       INT     0       6
11. 9       LIT     0       1
12. 10      STO     0       3
13. 11      LOD     0       3
14. 12      LIT     0       10
15. 13      OPR     0       6(LEQ)
16. 14      JPC     0       24
17. 15      CAL     0       2
18. 16      LOD     0       4
19. 17      OPR     0       14(WRITE)
20. 18      OPR     0       15(LINE)
21. 19      LOD     0       3
22. 20      LIT     0       1
23. 21      OPR     0       2(ADD)
24. 22      STO     0       3
25. 23      JMP     0       11
26. 24      OPR     0       0(RET)
27.
28. Process finished with exit code 0
```

7.5.4 PL0_code3.in 中间代码生成结果

代码 7-28 PL0_code3.in 中间代码生成结果

```
1.  C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users
    \TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code3.in
2.  0       JMP     0       51
3.  1       JMP     0       2
4.  2       INT     0       4
5.  3       LIT     0       1
6.  4       STO     1       4
7.  5       LIT     0       2
8.  6       STO     0       3
9.  7       LOD     0       3
10. 8       LOD     1       3
11. 9       OPR     0       7(LS)
```

```
12. 10      JPC     0       28
13. 11      LOD     1       3
14. 12      LOD     0       3
15. 13      OPR     0       5(DIV)
16. 14      LOD     0       3
17. 15      OPR     0       4(MUL)
18. 16      LOD     1       3
19. 17      OPR     0       8(EQU)
20. 18      JPC     0       23
21. 19      LIT     0       0
22. 20      STO     1       4
23. 21      LOD     1       3
24. 22      STO     0       3
25. 23      LOD     0       3
26. 24      LIT     0       1
27. 25      OPR     0       2(ADD)
28. 26      STO     0       3
29. 27      JMP     0       7
30. 28      OPR     0       0(RET)
31. 29      JMP     0       30
32. 30      INT     0       3
33. 31      LIT     0       2
34. 32      STO     1       3
35. 33      LOD     1       3
36. 34      LIT     0       100
37. 35      OPR     0       7(LS)
38. 36      JPC     0       50
39. 37      CAL     1       2
40. 38      LOD     1       4
41. 39      LIT     0       1
42. 40      OPR     0       8(EQU)
43. 41      JPC     0       45
44. 42      LOD     1       3
45. 43      OPR     0       14(WRITE)
46. 44      OPR     0       15(LINE)
47. 45      LOD     1       3
48. 46      LIT     0       1
49. 47      OPR     0       2(ADD)
50. 48      STO     1       3
51. 49      JMP     0       33
52. 50      OPR     0       0(RET)
53. 51      INT     0       7
54. 52      CAL     0       30
55. 53      OPR     0       0(RET)
56.
57. Process finished with exit code 0
```

7.5.5 PL0_code4.in 中间代码生成结果

代码 7-29 PL0_code4.in 中间代码生成结果

```
1.  C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users
    \TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code4.in
2.  0       JMP     0       101
3.  1       JMP     0       2
4.  2       INT     0       5
5.  3       LOD     1       3
6.  4       STO     0       3
7.  5       LOD     1       4
8.  6       STO     0       4
9.  7       LOD     1       3
10. 8       LOD     1       4
11. 9       OPR     0       4(MUL)
```

```
12.  10      STO     1       5
13.  11      OPR     0       0(RET)
14.  12      JMP     0       13
15.  13      INT     0       4
16.  14      LOD     1       3
17.  15      STO     1       7
18.  16      LIT     0       0
19.  17      STO     1       6
20.  18      LOD     1       4
21.  19      STO     0       3
22.  20      LOD     0       3
23.  21      LOD     1       7
24.  22      OPR     0       6(LEQ)
25.  23      JPC     0       29
26.  24      LIT     0       2
27.  25      LOD     0       3
28.  26      OPR     0       4(MUL)
29.  27      STO     0       3
30.  28      JMP     0       20
31.  29      LOD     0       3
32.  30      LOD     1       4
33.  31      OPR     0       10(GT)
34.  32      JPC     0       54
35.  33      LIT     0       2
36.  34      LOD     1       6
37.  35      OPR     0       4(MUL)
38.  36      STO     1       6
39.  37      LOD     0       3
40.  38      LIT     0       2
41.  39      OPR     0       5(DIV)
42.  40      STO     0       3
43.  41      LOD     0       3
44.  42      LOD     1       7
45.  43      OPR     0       6(LEQ)
46.  44      JPC     0       53
47.  45      LOD     1       7
48.  46      LOD     0       3
49.  47      OPR     0       3(SUB)
50.  48      STO     1       7
51.  49      LOD     1       6
52.  50      LIT     0       1
53.  51      OPR     0       2(ADD)
54.  52      STO     1       6
55.  53      JMP     0       29
56.  54      OPR     0       0(RET)
57.  55      JMP     0       56
58.  56      INT     0       4
59.  57      LOD     1       3
60.  58      STO     1       9
61.  59      LOD     1       4
62.  60      STO     0       3
63.  61      LOD     1       9
64.  62      LOD     0       3
65.  63      OPR     0       9(NEQ)
66.  64      JPC     0       82
67.  65      LOD     1       9
68.  66      LOD     0       3
69.  67      OPR     0       7(LS)
70.  68      JPC     0       73
71.  69      LOD     0       3
72.  70      LOD     1       9
73.  71      OPR     0       3(SUB)
74.  72      STO     0       3
75.  73      LOD     0       3
76.  74      LOD     1       9
77.  75      OPR     0       7(LS)
```

```
78.  76      JPC     0       81
79.  77      LOD     1       9
80.  78      LOD     0       3
81.  79      OPR     0       3(SUB)
82.  80      STO     1       9
83.  81      JMP     0       61
84.  82      LOD     1       9
85.  83      STO     1       5
86.  84      OPR     0       0(RET)
87.  85      JMP     0       86
88.  86      INT     0       3
89.  87      LOD     1       8
90.  88      LIT     0       1
91.  89      OPR     0       10(GT)
92.  90      JPC     0       100
93.  91      LOD     1       8
94.  92      LOD     1       9
95.  93      OPR     0       4(MUL)
96.  94      STO     1       9
97.  95      LOD     1       8
98.  96      LIT     0       1
99.  97      OPR     0       3(SUB)
100. 98      STO     1       8
101. 99      CAL     1       86
102. 100     OPR     0       0(RET)
103. 101     INT     0       14
104. 102     OPR     0       16(READ)
105. 103     STO     0       3
106. 104     OPR     0       16(READ)
107. 105     STO     0       4
108. 106     CAL     0       2
109. 107     LOD     0       5
110. 108     OPR     0       14(WRITE)
111. 109     OPR     0       15(LINE)
112. 110     OPR     0       16(READ)
113. 111     STO     0       3
114. 112     OPR     0       16(READ)
115. 113     STO     0       4
116. 114     CAL     0       13
117. 115     LOD     0       6
118. 116     OPR     0       14(WRITE)
119. 117     OPR     0       15(LINE)
120. 118     LOD     0       7
121. 119     OPR     0       14(WRITE)
122. 120     OPR     0       15(LINE)
123. 121     OPR     0       16(READ)
124. 122     STO     0       3
125. 123     OPR     0       16(READ)
126. 124     STO     0       4
127. 125     CAL     0       56
128. 126     LOD     0       5
129. 127     OPR     0       14(WRITE)
130. 128     OPR     0       15(LINE)
131. 129     OPR     0       16(READ)
132. 130     STO     0       8
133. 131     LIT     0       1
134. 132     STO     0       9
135. 133     CAL     0       86
136. 134     LOD     0       9
137. 135     OPR     0       14(WRITE)
138. 136     OPR     0       15(LINE)
139. 137     OPR     0       0(RET)
140.
141. Process finished with exit code 0
```

7.5.6 PL0_code5.in 中间代码生成结果

代码 7-30 PL0_code5.in 中间代码生成结果

```
1.  C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users
    \TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code5.in
2.  0       JMP     0       69
3.  1       JMP     0       2
4.  2       INT     0       4
5.  3       LIT     0       2
6.  4       STO     0       3
7.  5       LOD     0       3
8.  6       LIT     0       2
9.  7       OPR     0       5(DIV)
10. 8       LIT     0       2
11. 9       OPR     0       2(ADD)
12. 10      LIT     0       1
13. 11      OPR     0       3(SUB)
14. 12      OPR     0       14(WRITE)
15. 13      OPR     0       15(LINE)
16. 14      LOD     1       5
17. 15      LIT     0       0
18. 16      OPR     0       9(NEQ)
19. 17      JPC     0       33
20. 18      LOD     1       3
21. 19      LOD     1       4
22. 20      OPR     0       5(DIV)
23. 21      STO     1       6
24. 22      LOD     1       3
25. 23      LOD     1       6
26. 24      LOD     1       4
27. 25      OPR     0       4(MUL)
28. 26      OPR     0       3(SUB)
29. 27      STO     1       5
30. 28      LOD     1       4
31. 29      STO     1       3
32. 30      LOD     1       5
33. 31      STO     1       4
34. 32      JMP     0       14
35. 33      OPR     0       0(RET)
36. 34      JMP     0       51
37. 35      JMP     0       36
38. 36      INT     0       3
39. 37      LOD     2       3
40. 38      LIT     0       1
41. 39      OPR     0       9(NEQ)
42. 40      JPC     0       50
43. 41      LOD     2       3
44. 42      LIT     0       1
45. 43      OPR     0       3(SUB)
46. 44      STO     2       3
47. 45      LOD     2       4
48. 46      LOD     2       3
49. 47      OPR     0       4(MUL)
50. 48      STO     2       4
51. 49      CAL     1       36
52. 50      OPR     0       0(RET)
53. 51      INT     0       4
54. 52      CAL     0       36
55. 53      LIT     0       2
56. 54      OPR     0       6(LEQ)
57. 55      JPC     0       59
58. 56      LIT     0       2
59. 57      OPR     0       14(WRITE)
```

```
60. 58      OPR     0       15(LINE)
61. 59      LIT     0       2
62. 60      LIT     0       2
63. 61      OPR     0       8(EQU)
64. 62      JPC     0       68
65. 63      LIT     0       2
66. 64      LIT     0       1
67. 65      OPR     0       2(ADD)
68. 66      OPR     0       14(WRITE)
69. 67      OPR     0       15(LINE)
70. 68      OPR     0       0(RET)
71. 69      INT     0       9
72. 70      OPR     0       16(READ)
73. 71      STO     0       3
74. 72      OPR     0       16(READ)
75. 73      STO     0       4
76. 74      LOD     0       3
77. 75      LOD     0       4
78. 76      OPR     0       7(LS)
79. 77      JPC     0       84
80. 78      LOD     0       3
81. 79      STO     0       5
82. 80      LOD     0       4
83. 81      STO     0       3
84. 82      LOD     0       5
85. 83      STO     0       4
86. 84      LIT     0       1
87. 85      STO     0       5
88. 86      CAL     0       2
89. 87      LIT     0       2
90. 88      OPR     0       14(WRITE)
91. 89      LIT     0       2
92. 90      LOD     0       3
93. 91      OPR     0       4(MUL)
94. 92      OPR     0       14(WRITE)
95. 93      LIT     0       2
96. 94      LOD     0       3
97. 95      OPR     0       4(MUL)
98. 96      LIT     0       2
99. 97      OPR     0       5(DIV)
100. 98     OPR     0       14(WRITE)
101. 99     OPR     0       15(LINE)
102. 100    OPR     0       16(READ)
103. 101    STO     0       3
104. 102    LOD     0       3
105. 103    STO     0       4
106. 104    CAL     0       51
107. 105    LOD     0       4
108. 106    OPR     0       14(WRITE)
109. 107    OPR     0       15(LINE)
110. 108    OPR     0       0(RET)
111.
112. Process finished with exit code 0
```

7.6 解释执行结果

7.6.1 PL0_code0.in 解释执行结果

代码 7-31 PL0_code0.in 解释执行结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code0.in
Please Input:1
 Please Input:2
 1   2       0

Process finished with exit code 0
```

7.6.2 PL0_code1.in 解释执行结果

代码 7-32 PL0_code1.in 解释执行结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code1.in
Please Input:1
 22
Please Input:2
 24
Please Input:3
 26
Please Input:4
 28
Please Input:5
 30
Please Input:6
 32
Please Input:7
 34
Please Input:0

Process finished with exit code 0
```

7.6.3 PL0_code2.in 解释执行结果

代码 7-33 PL0_code2.in 解释执行结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code2.in
1
4
9
16
25
36
49
64
81
100

Process finished with exit code 0
```

7.6.4 PL0_code3.in 解释执行结果

代码 7-34 PL0_code3.in 解释执行结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code3.in
```

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97

Process finished with exit code 0
```

7.6.5 PL0_code4.in 解释执行结果

代码 7-35 PL0_code4.in 解释执行结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code4.in
Please Input:1
 Please Input:2
 2
Please Input:3
 Please Input:4
 0
3
Please Input:5
 Please Input:6
 1
Please Input:7
 5040

Process finished with exit code 0
```

7.6.6 PL0_code5.in 解释执行结果

代码 7-36 PL0_code5.in 解释执行结果

```
C:\Users\tttt\Desktop\tempProjects\PL0-Compiler\cmake-build-debug\PL0_Compiler.exe C:\Users\TTTT\Desktop\tempProjects\PL0-Compiler\test\PL0_code5.in
Please Input:1
 Please Input:2
 2
2       2       1
Please Input:3
 3
```

```
6

Process finished with exit code 0
```