编译原理实验报告

姓名： 张童

学号： 201600262022

班级： 16 级泰山学堂计算机取向

指导老师：王丽荣

目录

一、综述

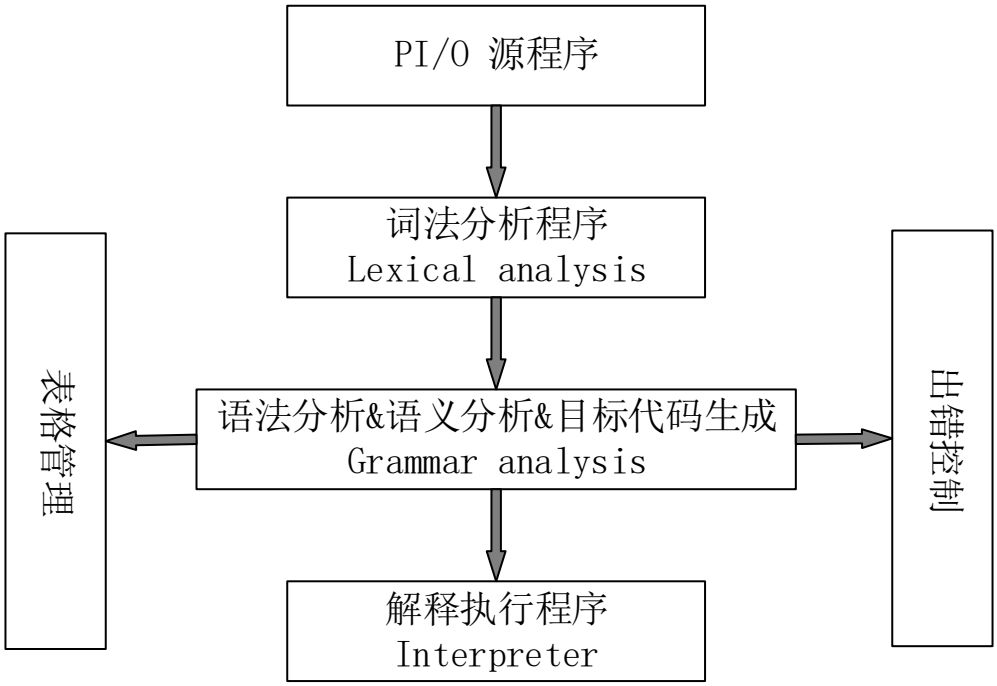
编译器（compiler），是一种计算机程序，它会将用某种编程语言写成的源代码（原始语言），转换成另一种编程语言（目标语言）。

它主要的目的是将便于人编写、阅读、维护的高级计算机语言所写作的源代码程序，翻译为计算机能解读、运行的低阶机器语言的程序，也就是可执行文件。编译器将原始程序（source program）作为输入，翻译产生使用目标语言（target language）的等价程序。

本实验通过对 PI/O 文法实现简单的编译器，加深我们队编译过程的理解。

二、实验部分

基本流程

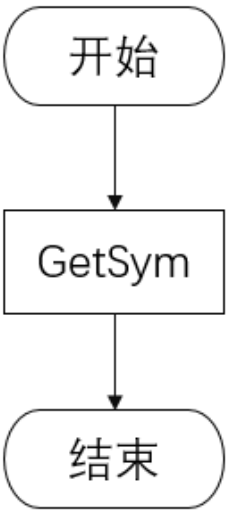


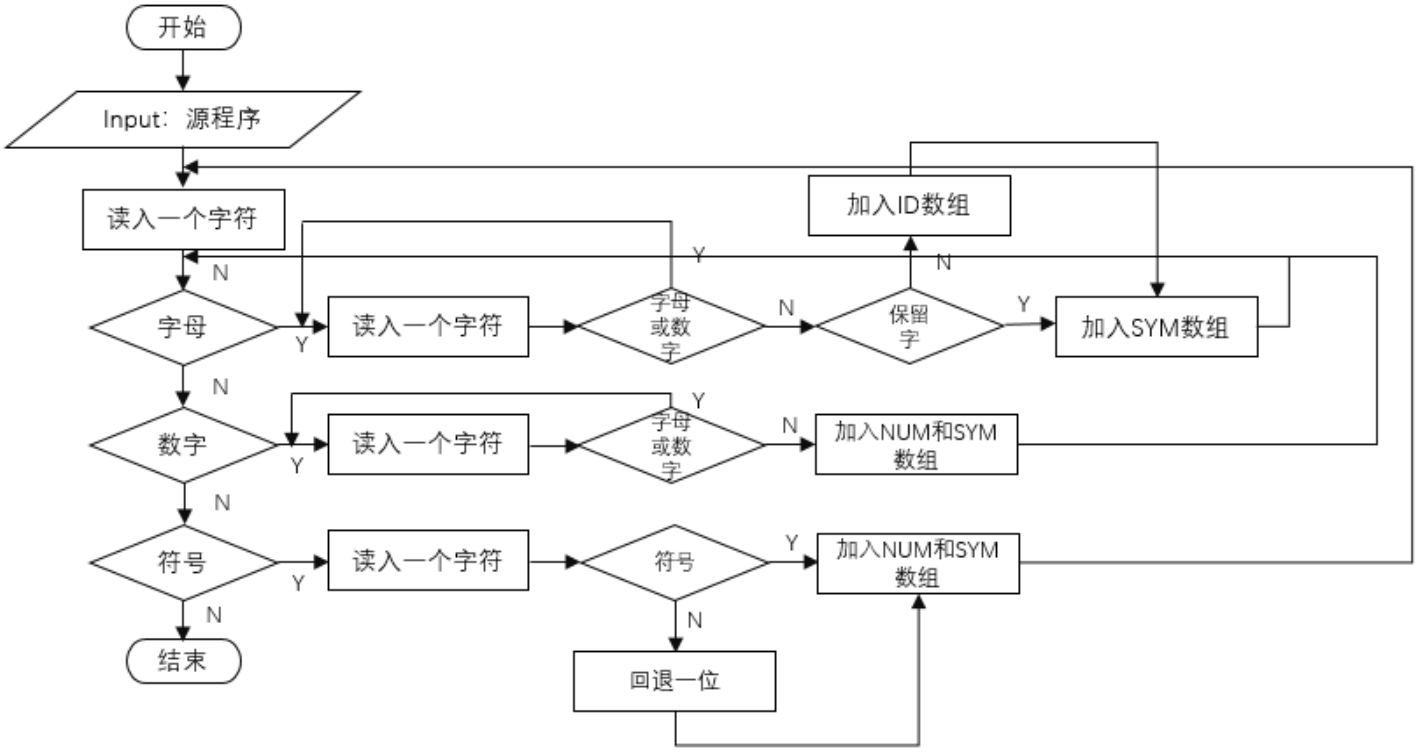
通过三个类实现 lexical analysis　grammar analysis　Interpreter

词法分析

Lexical analysis 类

基本流程





输入： 源程序

输出： 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。

完成的任务：

1. 滤掉单词间的空格。

2. 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。

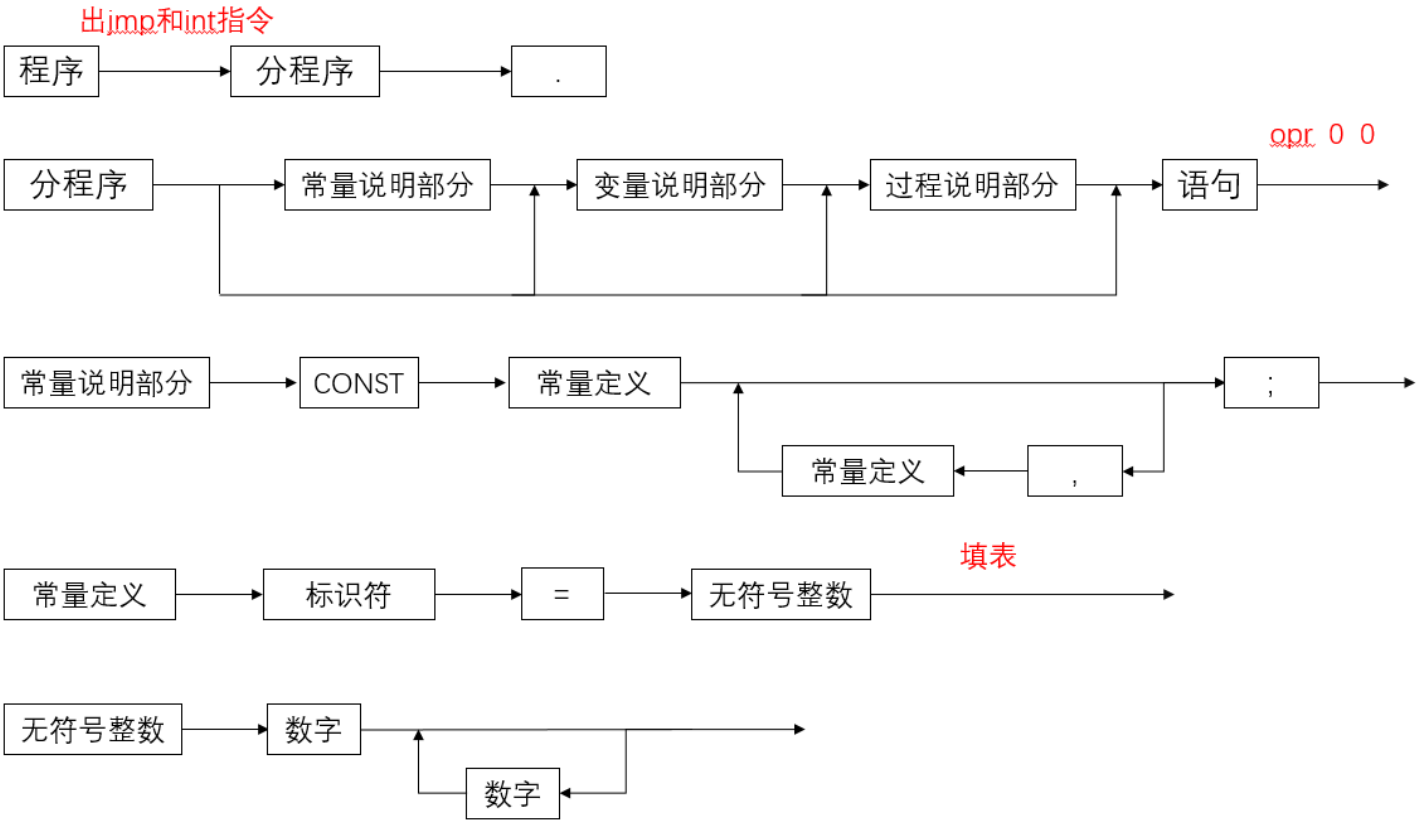
3. 识别标识符，标识符的类别为 0,0 放在 SYM 中，标识符本身的值放在 ID 中。

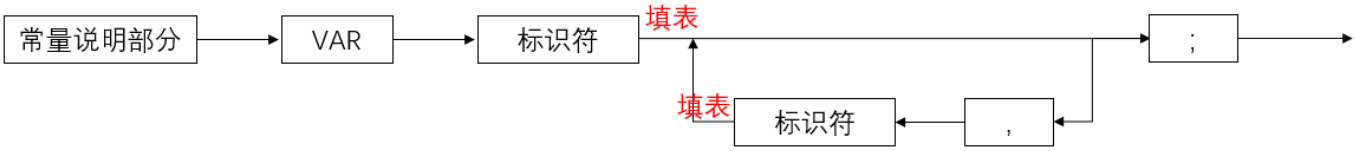
4. 拼数，将数的类别 30 放在 SYM 中，数本身的值放在 NUM 中。

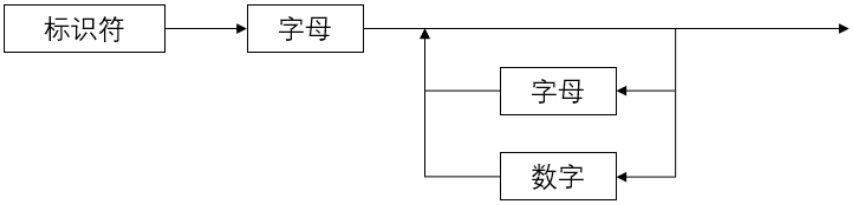
5. 拼由两个字符组成的运算符，如：>=、<=等等，识别后将类别存放在 SYM 中。

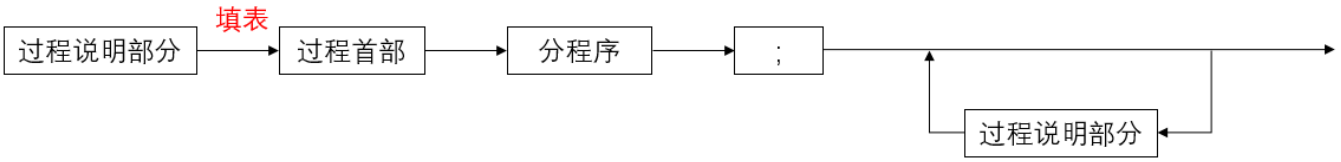
语法分析

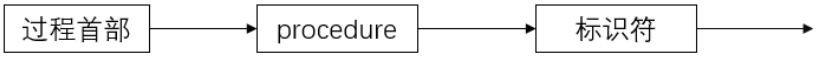
grammar analysis 类

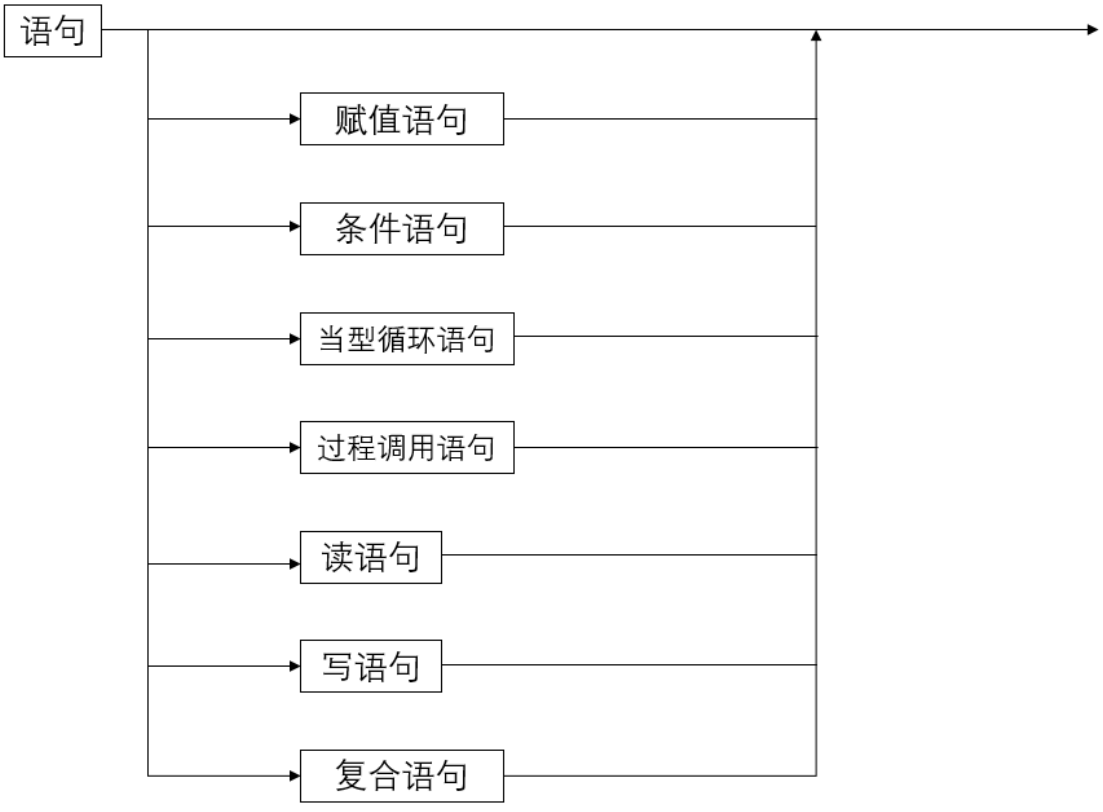


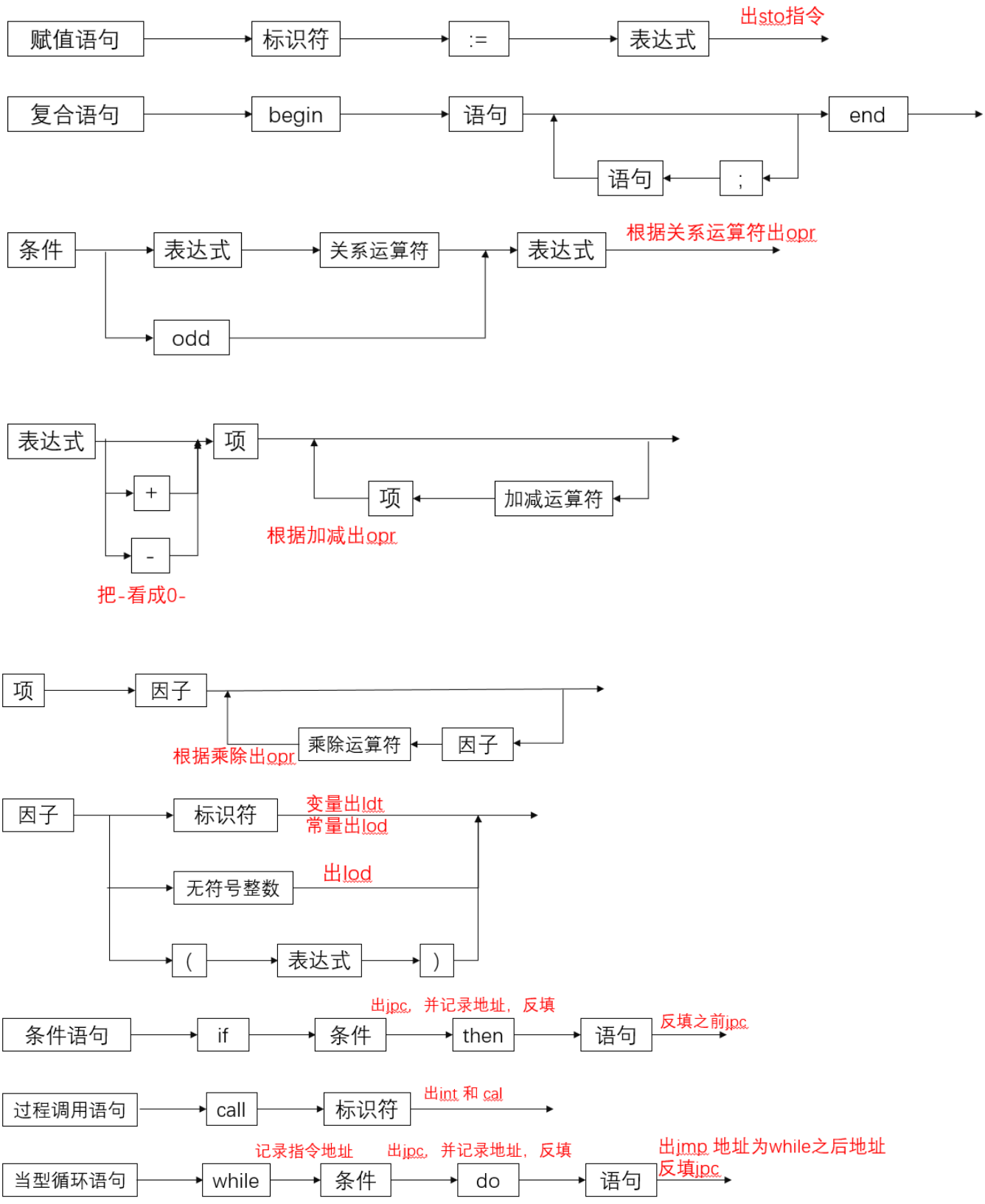

常量说明部分
VAR
标识符
填表
;
填表
标识符
,

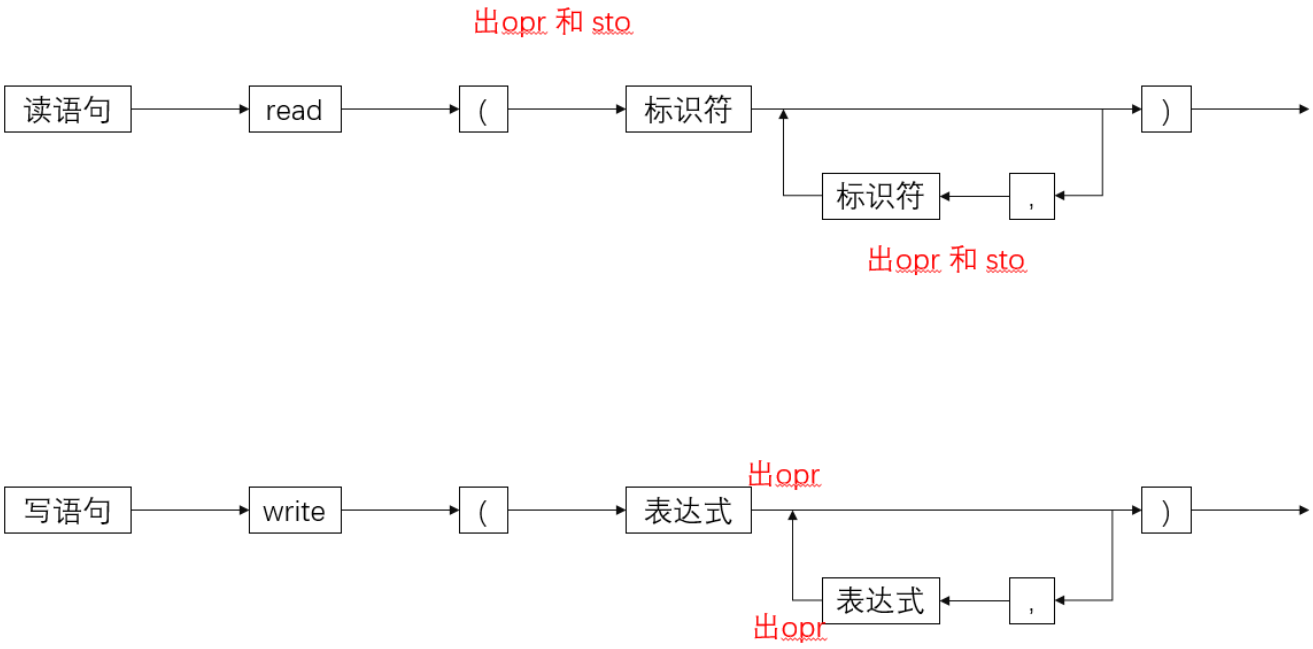

标识符
字母
字母
数字


过程说明部分
填表
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


赋值语句
标识符
:=
表达式
出sto指令
复合语句
begin
语句
end
语句
;
条件
表达式
关系运算符
表达式
根据关系运算符出opr
odd
表达式
项
+
-
项
加减运算符
根据加减出opr
把-看成0-
项
因子
乘除运算符
因子
根据乘除出opr
因子
标识符
变量出ldt
常量出lod
无符号整数
出lod
(
表达式
)
条件语句
if
条件
then
语句
出jpc，并记录地址，反填
反填之前jpc
过程调用语句
call
标识符
出int 和 cal
当型循环语句
while
条件
do
语句
记录指令地址
出jpc，并记录地址，反填
出jmp 地址为while之后地址
反填jpc



输入：lexical analysis 得到的 SYM ID NUM

输出：语法生成树

利用递归下降法

程序自动生成 image.gv(见附录)脚本，用 graphviz 生成树，imagefinal.png。

语义分析及目标代码生成

grammar analysis 类

输入：lexical analysis 得到的 SYM ID NUM

输出：目标代码

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有在 grammar analysis 中完成。工作分为两步：

a）建表

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 1 层，在主程序中定义的过程是 2 层，随着嵌套的深度增而层次数增大。PL/0 最多允许 4 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，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：一个过程的说明部分为：

```
const a=35,b=49;
 var c,d,e;
 procedure p;
   var g;
```

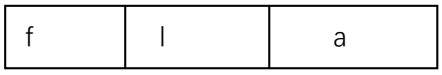
对它的常量、变量和过程说明处理后，TABLE 表中的信息如下：

NAME: c1	KIND: CONSTANT	LEVEL: 1	VAL:2
NAME: v1	KIND: VARIABLE	LEVEL: 1	VAL:3
NAME: v2	KIND: VARIABLE	LEVEL: 1	ADR: 4
NAME: v3	KIND: VARIABLE	LEVEL: 1	ADR: 5
NAME: v4	KIND: VAEIABLE	LEVEL: 1	ADR: 6
NAME: p1	KIND: PROCEDURE	LEVEL: 1	ADR: 2
NAME: v5	KIND: VARIABLE	LEVEL: 2	ADR: 3
NAME: p2	KIND: PROCEDURE	LEVEL: 1	ADR: 51
NAME: c2	KIND: CONSTANT	LEVEL: 2	ADR: 2
NAME: p3	KIND: PROCEDURE	LEVEL: 2	ADR: 36

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。每个过程的相对起始位置初值 DX=3。

B）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。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，其格式如下：



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,l 域的含义与 LOD 的相同。

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。

OPR 在 a 不同时的含义：

1 +

2 -

3 *

4 /

5 =

6 #

7 <

8 >

9 <=

10 >=

11 odd

12 read

13 write

解释执行

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．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系单元。

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）。

3．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。

1．INT　0　a

a:为局部量个数加 3

2．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。

3．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。

三、总结

通过实现一个实际编译器（PL/0 语言编译器），我对编译阶段（包括词法分析、语法分析、语义分析、中间代码生成等）和编译系统软件结构有了更深刻的理解。

附录：

生成的目标代码和表格:

```
0   c1    CONSTANT    1    2
1   v1    VARIABLE    1    3
2   v2    VARIABLE    1    4
3   v3    VARIABLE    1    5
4   v4    VARIABLE    1    6
5   p1    PROCEDURE    1    2
6   v5    VARIABLE    2    3
7   p2    PROCEDURE    1    51
8   c2    CONSTANT    2    2
9   p3    PROCEDURE    2    36
0   INT   0   4
1   JMP   0   67
2   LOD   1   3
3   OPR   0   13
4   LOD   1   4
5   OPR   0   13
6   LIT   0   2
7   STO   0   3
8   LOD   0   3
9   LIT   0   2
10   OPR   0   4
11   LIT   0   2
12   OPR   0   1
13   LIT   0   1
14   OPR   0   2
15   OPR   0   13
16   LOD   1   5
17   LIT   0   0
18   OPR   0   6
19   JPC   0   35
20   LOD   1   3
21   LOD   1   4
22   OPR   0   4
23   STO   1   6
24   LOD   1   3
25   LOD   1   6
26   LOD   1   4
27   OPR   0   3
28   OPR   0   2
```

```
40. 29   STO   1   5
41. 30   LOD   1   4
42. 31   STO   1   3
43. 32   LOD   1   5
44. 33   STO   1   4
45. 34   JMP   0   16
46. 35   OPR   0   0
47. 36   LOD   2   3
48. 37   LIT   0   1
49. 38   OPR   0   6
50. 39   JPC   0   50
51. 40   LOD   2   3
52. 41   LIT   0   1
53. 42   OPR   0   2
54. 43   STO   2   3
55. 44   LOD   2   4
56. 45   LOD   2   3
57. 46   OPR   0   3
58. 47   STO   2   4
59. 48   INT   0   0
60. 49   CAL   0   36
61. 50   OPR   0   0
62. 51   INT   0   0
63. 52   CAL   1   36
64. 53   LIT   0   2
65. 54   OPR   0   11
66. 55   JPC   0   58
67. 56   LIT   0   2
68. 57   OPR   0   13
69. 58   LIT   0   2
70. 59   LIT   0   2
71. 60   OPR   0   5
72. 61   JPC   0   66
73. 62   LIT   0   2
74. 63   LIT   0   1
75. 64   OPR   0   1
76. 65   OPR   0   13
77. 66   OPR   0   0
78. 67   OPR   0   12
79. 68   STO   0   3
80. 69   OPR   0   12
81. 70   STO   0   4
82. 71   LOD   0   3
83. 72   LOD   0   4
```

```
84. 73   OPR   0   7
85. 74   JPC   0   81
86. 75   LOD   0   3
87. 76   STO   0   5
88. 77   LOD   0   4
89. 78   STO   0   3
90. 79   LOD   0   5
91. 80   STO   0   4
92. 81   LIT   0   1
93. 82   STO   0   5
94. 83   INT   0   1
95. 84   CAL   1   2
96. 85   LIT   0   2
97. 86   OPR   0   13
98. 87   LIT   0   2
99. 88   LOD   0   3
100. 89   OPR   0   3
101. 90   OPR   0   13
102. 91   LIT   0   2
103. 92   LOD   0   3
104. 93   OPR   0   3
105. 94   LIT   0   2
106. 95   OPR   0   4
107. 96   OPR   0   13
108. 97   OPR   0   12
109. 98   STO   0   3
110. 99   LOD   0   3
111. 100   STO   0   4
112. 101   INT   0   0
113. 102   CAL   1   51
114. 103   LOD   0   4
115. 104   OPR   0   13
116. 105   OPR   0   0
```

Image.gv 文件:

```
1.  digraph tree{
2.  element0[label="program"]
3.  element1[label="subprogram"]
4.  element2[label="const description"]
5.  element3[label="const"]
6.  element4[label="const define"]
7.  element5[label="identifier"]
```

```
element6[label="="]
element7[label="unsigned int"]
element8[label=";"]
element9[label="variable description"]
element10[label="var"]
element11[label="identifier"]
element12[label=","]
element13[label="identifier"]
element14[label=","]
element15[label="identifier"]
element16[label=","]
element17[label="identifier"]
element18[label=";"]
element19[label="progress description"]
element20[label="progress head"]
element21[label="procedure"]
element22[label="identifier"]
element23[label=";"]
element24[label="subprogram"]
element25[label="variable description"]
element26[label="var"]
element27[label="identifier"]
element28[label=";"]
element29[label="statement"]
element30[label="compound statement"]
element31[label="begin"]
element32[label="statement"]
element33[label="assignment statement"]
element34[label="identifier"]
element35[label=":="]
element36[label="expression"]
element37[label="item"]
element38[label="factor"]
element39[label="unsigned int"]
element40[label=";"]
element41[label="statement"]
element42[label="write statement"]
element43[label="write"]
element44[label="("]
element45[label="expression"]
element46[label="item"]
element47[label="factor"]
element48[label="identifier"]
element49[label="mul div operator"]
```

```
element50[label="factor"]
element51[label="unsigned int"]
element52[label="add sub operator"]
element53[label="item"]
element54[label="factor"]
element55[label="unsigned int"]
element56[label="add sub operator"]
element57[label="item"]
element58[label="factor"]
element59[label="unsigned int"]
element60[label=")"]
element61[label=";"]
element62[label="statement"]
element63[label="while type loop statement"]
element64[label="while"]
element65[label="condition"]
element66[label="expression"]
element67[label="item"]
element68[label="factor"]
element69[label="identifier"]
element70[label="relation operator"]
element71[label="expression"]
element72[label="item"]
element73[label="factor"]
element74[label="unsigned int"]
element75[label="then"]
element76[label="statement"]
element77[label="compound statement"]
element78[label="begin"]
element79[label="statement"]
element80[label="assignment statement"]
element81[label="identifier"]
element82[label=":="]
element83[label="expression"]
element84[label="item"]
element85[label="factor"]
element86[label="identifier"]
element87[label="mul div operator"]
element88[label="factor"]
element89[label="identifier"]
element90[label=";"]
element91[label="statement"]
element92[label="assignment statement"]
element93[label="identifier"]
```

```
element94[label=":="]
element95[label="expression"]
element96[label="item"]
element97[label="factor"]
element98[label="identifier"]
element99[label="add sub operator"]
element100[label="item"]
element101[label="factor"]
element102[label="identifier"]
element103[label="mul div operator"]
element104[label="factor"]
element105[label="identifier"]
element106[label=";"]
element107[label="statement"]
element108[label="assignment statement"]
element109[label="identifier"]
element110[label=":="]
element111[label="expression"]
element112[label="item"]
element113[label="factor"]
element114[label="identifier"]
element115[label=";"]
element116[label="statement"]
element117[label="assignment statement"]
element118[label="identifier"]
element119[label=":="]
element120[label="expression"]
element121[label="item"]
element122[label="factor"]
element123[label="identifier"]
element124[label=";"]
element125[label="statement"]
element126[label="empty"]
element127[label="end"]
element128[label=";"]
element129[label="statement"]
element130[label="empty"]
element131[label="end"]
element132[label=";"]
element133[label="statement"]
element134[label="compound statement"]
element135[label="begin"]
element136[label="statement"]
element137[label="empty"]
```

```
element138[label="end"]
element139[label="."]
element0 -> element139
element0 -> element1
element1 -> element133
element133 -> element134
element134 -> element138
element134 -> element136
element136 -> element137
element134 -> element135
element1 -> element19
element19 -> element132
element19 -> element24
element24 -> element29
element29 -> element30
element30 -> element131
element30 -> element129
element129 -> element130
element30 -> element128
element30 -> element62
element62 -> element63
element63 -> element76
element76 -> element77
element77 -> element127
element77 -> element125
element125 -> element126
element77 -> element124
element77 -> element116
element116 -> element117
element117 -> element120
element120 -> element121
element121 -> element122
element122 -> element123
element117 -> element119
element117 -> element118
element77 -> element115
element77 -> element107
element107 -> element108
element108 -> element111
element111 -> element112
element112 -> element113
element113 -> element114
element108 -> element110
element108 -> element109
```

```
element77 -> element106
element77 -> element91
element91 -> element92
element92 -> element95
element95 -> element100
element100 -> element104
element104 -> element105
element100 -> element103
element100 -> element101
element101 -> element102
element95 -> element99
element95 -> element96
element96 -> element97
element97 -> element98
element92 -> element94
element92 -> element93
element77 -> element90
element77 -> element79
element79 -> element80
element80 -> element83
element83 -> element84
element84 -> element88
element88 -> element89
element84 -> element87
element84 -> element85
element85 -> element86
element80 -> element82
element80 -> element81
element77 -> element78
element63 -> element75
element63 -> element65
element65 -> element71
element71 -> element72
element72 -> element73
element73 -> element74
element65 -> element70
element65 -> element66
element66 -> element67
element67 -> element68
element68 -> element69
element63 -> element64
element30 -> element61
element30 -> element41
element41 -> element42
```

```
element42 -> element60
element42 -> element45
element45 -> element57
element57 -> element58
element58 -> element59
element45 -> element56
element45 -> element53
element53 -> element54
element54 -> element55
element45 -> element52
element45 -> element46
element46 -> element50
element50 -> element51
element46 -> element49
element46 -> element47
element47 -> element48
element42 -> element44
element42 -> element43
element30 -> element40
element30 -> element32
element32 -> element33
element33 -> element36
element36 -> element37
element37 -> element38
element38 -> element39
element33 -> element35
element33 -> element34
element30 -> element31
element24 -> element25
element25 -> element28
element25 -> element27
element25 -> element26
element19 -> element20
element20 -> element23
element20 -> element22
element20 -> element21
element1 -> element9
element9 -> element18
element9 -> element17
element9 -> element16
element9 -> element15
element9 -> element14
element9 -> element13
element9 -> element12
```

```
element9 -> element11
element9 -> element10
element1 -> element2
element2 -> element8
element2 -> element4
element4 -> element7
element4 -> element6
element4 -> element5
element2 -> element3
}
```

代码

头文件:

Lexical analysis.h

```
#ifndef LEXICAL_ANALYSIS_H_INCLUDED
#define LEXICAL_ANALYSIS_H_INCLUDED
using namespace std;
class lexical_analysis{
public:
    lexical_analysis(vector<pair<int, int> >& sym, vector<string>& id, vecto
r<string>& num);
private:
    string reserved_word[13] = {"const","var","procedure","begin","end","odd
","if","then","call","while","do","read","write"};
    // 算符&&界符
    string symbol[16] = {"+","-
","*","/",":=","=","#","<",">","<=",">=","(",")",";",",","."};
    vector<pair<int, int> > sym;
    vector<string> sym_output;
    vector<string> id;
    vector<string> num;
    void GetSym(string input_str);
    int Reserve(string word);
    int ReserveSymbol(string word);
    int get_length(int num);
    bool IsSymbol(char ch);
    void display();

};

```

```
#endif // LEXICAL_ANALYSIS_H_INCLUDED
```

grammar analysis.h

```
#ifndef GRAMMAR_ANALYSIS_H_INCLUDED
#define GRAMMAR_ANALYSIS_H_INCLUDED
#include<vector>
#include<strstream>
using namespace std;
struct node{
    string element;
    vector<node*> children;
    node* parent;
    int nodeId;
    node(string key,int node_id_input){
        element = key;
        nodeId = node_id_input;
    }
};
struct tableElement{
    string name;
    string kind;
    int level;
    int adr_or_value;
    tableElement(string NAME, string KIND,int LEVEL, int ADR_OR_VALUE){
        name = NAME;
        kind = KIND;
        level = LEVEL;
        adr_or_value = ADR_OR_VALUE;
    }
};
struct code{
    string f;
    int l;
    int a;
    code(string F,int L, int A){
        f = F;
        l = L;
        a = A;
    }
};
class grammar_analysis{
public:
```

```
40.     grammar_analysis(vector<pair<int, int> >sym, vector<string>id, vector<st
    ring> num, vector<code>& codeRepository);
41. private:
42.     node* root;
43.     node* pos_pointer;
44.     vector<pair<int, int> > sym;
45.     vector<string> id;
46.     vector<int> num;
47.     vector<node*> allNode;
48.     vector<tableElement> table;
49.     vector<code> codeRepo;
50.     int dx;
51.     int lev;
52.     int nodeIdCounter;
53.     int counter=0;
54.     int program();// first={const,var,procedure,[a-
    z],if,while,call,read,write,begin,Пе}
55.     int subprogram(int procedureAddr);// first={const,var,procedure,[a-
    z],if,while,call,read,write,begin,Пе}
56.     int const_description();// first={const}
57.     int const_define();// first={[a-z]}
58.     int unsigned_int();// first={[0-9]}
59.     int variable_description();// first={var}
60.     int identifier();// first={[a-z]}
61.     int progress_description();// first={procedure}
62.     int progress_head();// first={procedure}
63.     int statement();// first={[a-z],if,while,call,read,write,begin,Пе}
64.     int assignment_statement();// first={[a-z]}
65.     int compound_statement();// first={begin}
66.     int condition();// first={odd,+,-,[a-z],[0-9],(}
67.     int expression();// first={+,-,[a-z],[0-9],(}
68.     int item();// first={[a-z],[0-9],(}
69.     int factor();// first={[a-z],[0-9],(}
70.     int add_sub_operator();// first={+,-}
71.     int mul_div_operator();// first={*,/}
72.     int relation_operator();// first={=,#,<,<=,>,>=}
73.     int conditional_statement();// first={if}
74.     int call_statement();// first={call}
75.     int when_type_loop_statement();// first={while}
76.     int read_statement();// first={read}
77.     int write_statement();// first={write}
78.     //int letter();// first={[a-z]}
79.     //int digit();// first={[0,9]}
80.     int convertStringToInt(const string &s);
```

```
    void drawTree(vector<node*> nodes, vector<int> nodeNum);
    void addNode(const string& key);
    void generateDot();
    int IsInTable(string entity);
    void show_object_code_and_table();

};

#endif // GRAMMAR_ANALYSIS_H_INCLUDED
```

Interpreter.h

```
#ifndef INTERPRETER_H_INCLUDED
#define INTERPRETER_H_INCLUDED
#include"grammar_analysis.h"
class interpreter{
public:
    interpreter(vector<code> codeRepo);
private:
    void execute(vector<code> codeRepo);
};


#endif // INTERPRETER_H_INCLUDED
```

源文件

Lexical analysis.cpp

```
#include<iostream>
#include<vector>
#include<fstream>
#include"lexical_analysis.h"
using namespace std;
#define NUMBERID 30 // 数字的内部编码表示
lexical_analysis::lexical_analysis(vector<pair<int, int> >& sym, vector<string>& id, vector<string>& num){
    ifstream in("program.txt");
    string line;
    while(getline(in,line)){
        GetSym(line);
    }
    in.close();
```

```
    sym = this->sym;
    id = this->id;
    num = this->num;
    //display();
}
int lexical_analysis::Reserve(string word){//判断一个词是不是保留字
    int code_num = 0;
    int count_num = 0;
    for(int i=0;i<13;i++){
        count_num++;
        if(word == reserved_word[i]){
            code_num = count_num;
            return code_num;
        }
    }
    return code_num;
}

int lexical_analysis::ReserveSymbol(string word){//判断是不是符号，是的话返回符号的编号
    int code_num = 13;
    for(int i=0;i<16;i++){
        code_num++;
        if(word == symbol[i]){
            return code_num;
        }
    }
    return 0;
}

bool lexical_analysis::IsSymbol(char ch){//判断是不是符号
    for(int i=0;i<16;i++){
        if(ch==symbol[i][0]){
            return 1;
        }
    }
    return 0;
}

void lexical_analysis::GetSym(string input_str){
int index = 0;
while(index<input_str.length()){
    int code;
    int value=-1;
```

```
57.     string strToken = "";
58.     while(index<input_str.length()&&input_str[index]==' '){
59.         index++;
60.     }
61.     if(index>=input_str.length()){
62.         return;
63.     }
64.     if(islower(input_str[index])){
65.         strToken+=input_str[index];
66.         index++;
67.         while(index<input_str.length()&&(islower(input_str[index])||isdigit(
    input_str[index]))){
68.             strToken+=input_str[index];
69.             index++;
70.         }
71.         code = Reserve(strToken);
72.         // value = -1;
73.         if(code==0){
74.             value = id.size();
75.             id.push_back(strToken);
76.         }
77.         sym.push_back(make_pair(code,value));
78.         sym_output.push_back(strToken);
79.     }
80.     else if(isdigit(input_str[index])){
81.         strToken+=input_str[index];
82.         index++;
83.         while(index<input_str.length()&&(isdigit(input_str[index]))){
84.             strToken+=input_str[index];
85.             index++;
86.         }
87.         code = NUMBERID;
88.         value = num.size();
89.         num.push_back(strToken);
90.         sym.push_back(make_pair(code,value));
91.         sym_output.push_back(strToken);
92.     }
93.     else if(IsSymbol(input_str[index])){
94.         strToken += input_str[index];
95.         index++;
96.         if(strToken=="<"||strToken==">"){
97.             if(index<input_str.length()&&input_str[index]=='='){
98.                 strToken+="=";
99.                 index++;
```

```
            }
        }
        if(strToken==":"){
            if(index<input_str.length()&&input_str[index]=='='){
                strToken+="=";
                index++;
            }
            else{
                cout<<"ERROR:"<<input_str<<endl;
                return;
            }
        }
        code = ReserveSymbol(strToken);
        sym.push_back(make_pair(code,value));
        sym_output.push_back(strToken);
    }
    else{
        cout<<"ERROR:"<<input_str<<endl;
        return;
    }
}


}

int lexical_analysis::get_length(int num){//计算数字占的位数
    if(num==0){
        return 1;
    }
    else{
        int length = 0;
        if(num<0){
            num = -num;
            length++;
        }
        while(num>0){
        num/=10;
        length++;
        }
        return length;
    }

}
void lexical_analysis::display(){
```

```
    for(int i=0;i<sym.size();i++){
        cout<<i+1<<'('<<sym[i].first<<' '<<sym[i].second<<')'<<endl;
    }
    for(int i=0;i<id.size();i++){
        cout<<id[i]<<endl;
    }
    cout<<endl;
    for(int i=0;i<num.size();i++){
        cout<<num[i]<<endl;
    }
    int first_length = 11;
    int second_length =11;
    cout<<"  symbol     class      address"<<endl;
    for(int i =0;i<sym_output.size();i++){
        cout<<i+1<<" ";
        string str1_temp = "";
        string str2_temp = "";
        for(int j=0;j<first_length-sym_output[i].size();j++){str1_temp+=" ";}
        for(int j=0;j<second_length-get_length(sym[i].first);j++){str2_temp+=" ";}
        cout<<sym_output[i]<<str1_temp<<sym[i].first<<str2_temp;
        if(sym[i].first==0){
            cout<<sym[i].second<<endl;
        }
        else if(sym[i].first==30){
            cout<<sym[i].second<<endl;
        }
        else{
            cout<<endl;
            continue;
        }
    }
}
```

grammar analysis.cpp

```
#include<iostream>
#include<vector>
#include<assert.h>
#include<fstream>
#include<stack>
#include<string>
```

```
7.  #include "grammar_analysis.h"
8.  using namespace std;
9.  #define PROGRAM "program"
10. #define SUBPROGRAM "subprogram"
11. #define CONST_DESCRIPTION "const description"
12. #define CONST_DEFINE "const define"
13. #define UNSIGNED_INT "unsigned int"
14. #define VARIABLE_DESCRIPTION "variable description"
15. #define IDENTIFIER "identifier"
16. #define PROGRESS_DESCRIPTION "progress description"
17. #define PROGRESS_HEAD "progress head"
18. #define STATEMENT "statement"
19. #define ASSIGNMENT_STATEMENT "assignment statement"
20. #define COMPOUND_STATEMENT "compound statement"
21. #define CONDITION "condition"
22. #define EXPRESSION "expression"
23. #define ITEM "item"
24. #define FACTOR "factor"
25. #define ADD_SUB_OPERATOR "add sub operator"
26. #define MUL_DIV_OPERATOR "mul div operator"
27. #define RELATION_OPERATOR "relation operator"
28. #define CONDITIONAL_STATEMENT "conditional statement"
29. #define CALL_STATEMENT "call statement"
30. #define WHEN_TYPE_LOOP_STATEMENT "while type loop statement"
31. #define READ_STATEMENT "read statement"
32. #define WRITE_STATEMENT "write statement"
33. grammar_analysis::grammar_analysis(vector<pair<int, int> >sym, vector<string
    >id, vector<string> num, vector<code>& codeRepository){
34.         this->sym = sym;
35.         this->id = id;
36.         for(int i=0;i<num.size();i++){
37.             this->num.push_back(convertStringToInt(num[i]));
38.         }
39.         dx = 3;
40.         lev = 0;
41.         nodeIdCounter = 0;
42.         root = new node(PROGRAM,nodeIdCounter);
43.         nodeIdCounter++;
44.         pos_pointer = root;
45.         root->parent = NULL;
46.         allNode.push_back(root);
47.         program();
48.         codeRepository = this->codeRepo;
49.         vector<node*> nodes;
```

```
50.         vector<int> nodeNum;
51.         nodes.push_back(root);
52.         nodeNum.push_back(root->children.size());
53.         //drawTree(nodes, nodeNum);
54.         //generateDot();
55.         show_object_code_and_table();
56. }
57. void grammar_analysis::addNode(const string& key){
58.     //cout<<key<<endl;
59.     node* childnode = new node(key,nodeIdCounter);
60.     nodeIdCounter++;
61.     childnode->parent = pos_pointer;
62.     pos_pointer->children.push_back(childnode);
63.     pos_pointer = childnode;
64.     allNode.push_back(childnode);
65. }
66. // 程序 first={const,var,procedure,[a-z],if,while,call,read,write,begin,空}
67. int grammar_analysis::program(){
68.     addNode(SUBPROGRAM);
69.     codeRepo.push_back(code("INT",0,-1));
70.     codeRepo.push_back(code("JMP",0,-1));
71.     int temp = subprogram(-1);
72.     if(temp==-1){
73.         return  -1;
74.     }
75.     pos_pointer = pos_pointer->parent;
76.     if(sym[counter].first==29){
77.         addNode(".");
78.         pos_pointer = pos_pointer->parent;
79.         counter++;
80.         return 1;
81.     }
82.     else{
83.         cout<<"Error in program description of "<<sym[counter].first<<endl;

84.         return -1;
85.     }
86.
87. }
88. // 分程序 first={const,var,procedure,[a-z],if,while,call,read,write,begin,
    空}
89. int grammar_analysis::subprogram(int procedureAddr){
90.     lev++;
91.     dx = 3;
```

```
92.     int temp;
93.     if(sym[counter].first==1){
94.         addNode(CONST_DESCRIPTION);
95.         temp = const_description();
96.         if(temp==-1){
97.             return -1;
98.         }
99.         pos_pointer = pos_pointer->parent;
100.     }
101.     if(sym[counter].first==2){
102.         addNode(VARIABLE_DESCRIPTION);
103.         temp = variable_description();
104.         if(temp==-1){
105.             return -1;
106.         }
107.         pos_pointer = pos_pointer->parent;
108.     }
109.     if(sym[counter].first==3){
110.         addNode(PROGRESS_DESCRIPTION);
111.         temp = progress_description();
112.         if(temp==-1){
113.             return -1;
114.         }
115.         pos_pointer = pos_pointer->parent;
116.     }
117.     addNode(STATEMENT);
118.     if(procedureAddr!=-1){
119.         table[procedureAddr].adr_or_value = codeRepo.size();
120.     }
121.     else{
122.         codeRepo[1].a = codeRepo.size();
123.         int intnum=0;
124.         for(int i =0;i<table.size();i++){
125.             if(table[i].kind=="PROCEDURE"){
126.                 codeRepo[0].a = intnum;
127.                 break;
128.             }
129.             if(table[i].kind=="VARIABLE"){
130.                 intnum++;
131.             }
132.         }
133.     }
134.     temp = statement();
135.     if(temp==-1){
```

```
136.         return -1;
137.     }
138.     codeRepo.push_back(code("OPR",0,0));
139.     pos_pointer = pos_pointer->parent;
140.     lev--;
141.     return 1;
142. }
143. // 常量说明部分 first={const}
144. int grammar_analysis::const_description(){
145.     int temp;
146.     if(sym[counter].first==1){
147.         addNode("const");
148.         pos_pointer = pos_pointer->parent;
149.         counter++;
150.     }
151.     else{
152.         cout<<"Error in const description of "<<sym[counter].first<<endl;
153.         return -1;
154.     }
155.     addNode(CONST_DEFINE);
156.     temp = const_define();
157.     if(temp==-1){
158.         return -1;
159.     }
160.     pos_pointer = pos_pointer->parent;
161.     while(sym[counter].first==28){
162.         addNode(",");
163.         pos_pointer = pos_pointer->parent;
164.         counter++;
165.         addNode(CONST_DEFINE);
166.         temp=const_define();
167.         if(temp==-1){
168.             return -1;
169.         }
170.         pos_pointer = pos_pointer->parent;
171.     }
172.     if(sym[counter].first==27){
173.         addNode(";");
174.         pos_pointer = pos_pointer->parent;
175.         counter++;
176.         return 1;
177.     }
178.     else{
179.         cout<<"Error in const description of "<<sym[counter].first<<endl;
```

```
180.         return -1;
181.     }
182. }
183. // 常量定义 first={[a-z]}
184. int grammar_analysis::const_define(){
185.     int temp;
186.     string tempname;
187.     int tempnum;
188.     if(sym[counter].first == 0){
189.         tempname = id[sym[counter].second];
190.         addNode(IDENTIFIER);
191.         pos_pointer = pos_pointer->parent;
192.         counter++;
193.     }
194.     else{
195.         cout<<"Error in const define of "<<sym[counter].first<<endl;
196.         return -1;
197.     }
198.     if(sym[counter].first == 19){
199.         addNode("=");
200.         pos_pointer = pos_pointer->parent;
201.         counter++;
202.     }
203.     else{
204.         cout<<"Error in const define of "<<sym[counter].first<<endl;
205.         return -1;
206.     }
207.     addNode(UNSIGNED_INT);
208.     temp = unsigned_int();
209.     pos_pointer = pos_pointer->parent;
210.     tempnum = num[sym[counter-1].second];
211.     table.push_back(tableElement(tempname,"CONSTANT",lev,tempnum));
212.     return temp;
213. }
214. // 无符号整数 first={[0-9]}
215. int grammar_analysis::unsigned_int(){
216.     if(sym[counter].first==30){
217.         counter++;
218.         return 1;
219.     }
220.     else{
221.         cout<<"Error in defining unsigned int "<<sym[counter].first<<endl;

222.         return -1;
```

```
223.     }
224. }
225. // 变量说明部分 first={var}
226. int grammar_analysis::variable_description(){
227.     int temp;
228.     if(sym[counter].first==2){
229.         addNode("var");
230.         pos_pointer = pos_pointer->parent;
231.         counter++;
232.     }
233.     else{
234.         cout<<"Error in variable description of "<<sym[counter].first<<endl
    ;
235.         return -1;
236.     }
237.     addNode(IDENTIFIER);
238.     temp = identifier();
239.     table.push_back(tableElement(id[sym[counter-
    1].second],"VARIABLE",lev,dx));
240.     dx++;
241.     if(temp==-1){
242.         return -1;
243.     }
244.     pos_pointer = pos_pointer->parent;
245.     while(sym[counter].first==28){
246.         addNode(",");
247.         pos_pointer = pos_pointer->parent;
248.         counter++;
249.         addNode(IDENTIFIER);
250.         temp=identifier();
251.         table.push_back(tableElement(id[sym[counter-
    1].second],"VARIABLE",lev,dx));
252.         dx++;
253.         if(temp==-1){
254.             return -1;
255.         }
256.         pos_pointer = pos_pointer->parent;
257.     }
258.     if(sym[counter].first==27){
259.         addNode(";");
260.         pos_pointer = pos_pointer->parent;
261.         counter++;
262.         return 1;
263.     }
```

```
264.     else{
265.         cout<<"Error in variavle description of "<<sym[counter].first<<endl
    ;
266.         return -1;
267.     }
268. }
269. // 标识符 first={[a-z]}
270. int grammar_analysis::identifier(){
271.     if(sym[counter].first==0){
272.         counter++;
273.         return 1;
274.     }
275.     else{
276.         cout<<"Error in defining identifier "<<sym[counter].first<<endl;
277.         return -1;
278.     }
279. }
280. // 过程说明部分 first={procedure}
281. int grammar_analysis::progress_description(){
282.     int temp;
283.     int procedureIndex;
284.     addNode(PROGRESS_HEAD);
285.     procedureIndex = table.size();
286.     table.push_back(tableElement(id[sym[counter+1].second],"PROCEDURE",lev,
    -1));
287.     temp = progress_head();
288.     if(temp==-1){
289.         return -1;
290.     }
291.     pos_pointer = pos_pointer->parent;
292.     addNode(SUBPROGRAM);
293.     temp = subprogram(procedureIndex);
294.     if(temp==-1){
295.         return -1;
296.     }
297.     pos_pointer = pos_pointer->parent;
298.     if(sym[counter].first==27){
299.         addNode(";");
300.         pos_pointer = pos_pointer->parent;
301.         counter++;
302.         temp = 1;
303.     }
304.     else{
```

```
305.         //cout<<"Error in progress description of"<<sym[counter].first<<end
    l;
306.         return -1;
307.     }
308.     if(sym[counter].first==3){// 针对此文法，此文法中 procedure 不可能出现在其
    他的地方
309.         addNode(PROGRESS_DESCRIPTION);
310.         temp = progress_description();
311.         pos_pointer = pos_pointer->parent;
312.         return temp;
313.     }
314.     else{
315.         return 1;
316.     }
317. }
318. // 过程首部 first={procedure}
319. int grammar_analysis::progress_head(){
320.     int temp;
321.     if(sym[counter].first==3){
322.         addNode("procedure");
323.         pos_pointer = pos_pointer->parent;
324.         counter++;
325.     }
326.     else{
327.         cout<<"Error in progress head of"<<sym[counter].first<<endl;
328.     }
329.     addNode(IDENTIFIER);
330.     temp = identifier();
331.     if(temp==-1){
332.         return -1;
333.     }
334.     pos_pointer = pos_pointer->parent;
335.     if(sym[counter].first==27){
336.         addNode(";");
337.         pos_pointer = pos_pointer->parent;
338.         counter++;
339.         return 1;
340.     }
341.     else{
342.         cout<<"Error in progress head of "<<sym[counter].first<<endl;
343.         return -1;
344.     }
345. }
346. // 语句 first={[a-z],if,while,call,read,write,begin,空}
```

```
347. int grammar_analysis::statement(){
348.     int temp;
349.     if(sym[counter].first==0){
350.         addNode(ASSIGNMENT_STATEMENT);
351.         temp = assignment_statement();
352.         pos_pointer = pos_pointer->parent;
353.         return 1;
354.     }
355.     else if(sym[counter].first==7){
356.         addNode(CONDITIONAL_STATEMENT);
357.         temp = conditional_statement();
358.         pos_pointer = pos_pointer->parent;
359.         return 1;
360.     }
361.     else if(sym[counter].first==10){
362.         addNode(WHEN_TYPE_LOOP_STATEMENT);
363.         temp = when_type_loop_statement();
364.         pos_pointer = pos_pointer->parent;
365.         return 1;
366.     }
367.     else if(sym[counter].first==9){
368.         addNode(CALL_STATEMENT);
369.         temp = call_statement();
370.         pos_pointer = pos_pointer->parent;
371.         return 1;
372.     }
373.     else if(sym[counter].first==12){
374.         addNode(READ_STATEMENT);
375.         temp = read_statement();
376.         pos_pointer = pos_pointer->parent;
377.         return 1;
378.     }
379.     else if(sym[counter].first==13){
380.         addNode(WRITE_STATEMENT);
381.         temp = write_statement();
382.         pos_pointer = pos_pointer->parent;
383.         return 1;
384.     }
385.     else if(sym[counter].first==4){
386.         addNode(COMPOUND_STATEMENT);
387.         temp = compound_statement();
388.         pos_pointer = pos_pointer->parent;
389.         return 1;
390.     }
```

```
391.     else{
392.         addNode("empty");
393.         pos_pointer = pos_pointer->parent;
394.         return 1;
395.     }
396. }
397. // 赋值语句 first={[a-z]}
398. int grammar_analysis::assignment_statement(){
399.     int indexOfVriable;
400.     if(sym[counter].first==0){
401.         indexOfVriable = IsInTable(id[sym[counter].second]);
402.         if(indexOfVriable==-1){
403.             cout<<id[sym[counter].second]<<"is used before statement"<<endl
    ;
404.             return -1;
405.         }
406.         addNode(IDENTIFIER);
407.         pos_pointer = pos_pointer->parent;
408.         counter++;
409.     }
410.     else{
411.         cout<<"Error in assignment statement of"<<sym[counter].first<<endl;

412.         return -1;
413.     }
414.     if(sym[counter].first==18){
415.         addNode(":=");
416.         pos_pointer = pos_pointer->parent;
417.         counter++;
418.     }
419.     else{
420.         cout<<"Error in assignment statement of"<<sym[counter].first<<endl;

421.         return -1;
422.     }
423.     addNode(EXPRESSION);
424.     int temp = expression();
425.     codeRepo.push_back(code("STO",lev-
    table[indexOfVriable].level,table[indexOfVriable].adr_or_value));
426.     pos_pointer = pos_pointer->parent;
427.     return temp;
428. }
429. // 复合语句 first={begin}
430. int grammar_analysis::compound_statement(){
```

```
431.     int temp;
432.     if(sym[counter].first==4){
433.         addNode("begin");
434.         pos_pointer = pos_pointer->parent;
435.         counter++;
436.     }
437.     else{
438.         cout<<"Error in compound statement of"<<sym[counter].first<<endl;
439.         return -1;
440.     }
441.     addNode(STATEMENT);
442.     temp = statement();
443.     if(temp == -1){
444.         return -1;
445.     }
446.     pos_pointer = pos_pointer->parent;
447.     while(sym[counter].first==27){
448.         addNode(";");
449.         pos_pointer = pos_pointer->parent;
450.         counter++;
451.         addNode(STATEMENT);
452.         temp=statement();
453.         if(temp==-1){
454.             return -1;
455.         }
456.         pos_pointer = pos_pointer->parent;
457.     }
458.     if(sym[counter].first==5){
459.         addNode("end");
460.         pos_pointer = pos_pointer->parent;
461.         counter++;
462.         return 1;
463.     }
464.     else{
465.         cout<<"Error in compound statement of"<<sym[counter].first<<endl;
466.         return -1;
467.     }
468. }
469. // 条件 first={odd,+,-,[a-z],[0-9],(}
470. int grammar_analysis::condition(){
471.     int temp;
472.     int tempsym;
473.     if(sym[counter].first==6){
474.         tempsym = 6;
```

```
475.         addNode("odd");
476.         pos_pointer = pos_pointer->parent;
477.         counter++;
478.         addNode(STATEMENT);
479.         temp = expression();
480.         codeRepo.push_back(code("OPR",0,11));
481.         pos_pointer = pos_pointer->parent;
482.         return temp;
483.     }
484.     else{
485.         addNode(EXPRESSION);
486.         temp = expression();
487.         if(temp==-1){
488.             return -1;
489.         }
490.         pos_pointer = pos_pointer->parent;
491.         addNode(RELATION_OPERATOR);
492.         tempsym = sym[counter].first;
493.         temp = relation_operator();
494.         if(temp==-1){
495.             return -1;
496.         }
497.         pos_pointer = pos_pointer->parent;
498.         addNode(EXPRESSION);
499.         temp = expression();
500.         pos_pointer = pos_pointer->parent;
501.         if(tempsym==19){
502.             codeRepo.push_back(code("OPR",0,5));
503.         }
504.         else if(tempsym==20){
505.             codeRepo.push_back(code("OPR",0,6));
506.         }
507.         else if(tempsym==21){
508.             codeRepo.push_back(code("OPR",0,7));
509.         }
510.         else if(tempsym==22){
511.             codeRepo.push_back(code("OPR",0,8));
512.         }
513.         else if(tempsym==23){
514.             codeRepo.push_back(code("OPR",0,9));
515.         }
516.         else if(tempsym==24){
517.             codeRepo.push_back(code("OPR",0,10));
518.         }
```

```
519.         return temp;
520.     }
521. }
522. // 表达式 first={+,-,[a-z],[0-9],(}
523. int grammar_analysis::expression(){
524.     int sign = 0;
525.     if(sym[counter].first==14||sym[counter].first==15){
526.         if(sym[counter].first==15){
527.             codeRepo.push_back(code("LIT",0,0));
528.             sign = 1;
529.         }
530.         addNode("[+|-]");
531.         pos_pointer = pos_pointer->parent;
532.         counter++;
533.     }
534.     int temp;
535.     addNode(ITEM);
536.     temp = item();
537.     if(temp==-1){
538.         return -1;
539.     }
540.     if(sign==1){
541.         codeRepo.push_back(code("OPR",0,2));
542.     }
543.     pos_pointer = pos_pointer->parent;
544.     while(sym[counter].first==14||sym[counter].first==15){
545.         int tempsym = sym[counter].first;
546.         addNode(ADD_SUB_OPERATOR);
547.         add_sub_operator();
548.         pos_pointer = pos_pointer->parent;
549.         addNode(ITEM);
550.         temp = item();
551.         if(temp==-1){
552.         return -1;
553.         }
554.         if(tempsym==14){
555.             codeRepo.push_back(code("OPR",0,1));
556.         }
557.         else if(tempsym==15){
558.             codeRepo.push_back(code("OPR",0,2));
559.         }
560.         pos_pointer = pos_pointer->parent;
561.     }
562.     return 1;
```

```
563. }
564. // 项 first={[a-z],[0-9],(}
565. int grammar_analysis::item(){
566.     int temp;
567.     addNode(FACTOR);
568.     temp = factor();
569.     if(temp==-1){
570.         return -1;
571.     }
572.     pos_pointer = pos_pointer->parent;
573.     while(sym[counter].first==16||sym[counter].first==17){
574.         int tempsym = sym[counter].first;
575.         addNode(MUL_DIV_OPERATOR);
576.         mul_div_operator();
577.         pos_pointer = pos_pointer->parent;
578.         addNode(FACTOR);
579.         temp = factor();
580.         if(temp==-1){
581.         return -1;
582.         }
583.         if(tempsym==16){
584.             codeRepo.push_back(code("OPR",0,3));
585.         }
586.         else if(tempsym==17){
587.             codeRepo.push_back(code("OPR",0,4));
588.         }
589.         pos_pointer = pos_pointer->parent;
590.     }
591.     return 1;
592. }
593. // 因子 first={[a-z],[0-9],(}
594. int grammar_analysis::factor(){
595.     int temp;
596.     if(sym[counter].first==0){
597.         addNode(IDENTIFIER);
598.         int tempindex = IsInTable(id[sym[counter].second]);
599.         //cout<<id[sym[counter].second]<<"  "<<tempindex<<endl;
600.         if(tempindex==-1){
601.             cout<<id[sym[counter].second]<<"is used before statement"<<endl
    ;
602.             return -1;
603.         }
604.         else if(table[tempindex].kind=="CONSTANT"){
```

```
605.             codeRepo.push_back(code("LIT",0,table[tempindex].adr_or_value))
    ;
606.         }
607.         else if(table[tempindex].kind=="VARIABLE"){
608.             codeRepo.push_back(code("LOD",lev-
    table[tempindex].level,table[tempindex].adr_or_value));
609.         }
610.         temp = identifier();
611.         pos_pointer = pos_pointer->parent;
612.         return temp;
613.     }
614.     else if(sym[counter].first==30){
615.         addNode(UNSIGNED_INT);
616.         codeRepo.push_back(code("LIT",0,num[sym[counter].second]));
617.         temp = unsigned_int();
618.         pos_pointer = pos_pointer->parent;
619.         return temp;
620.     }
621.     else if(sym[counter].first==25){
622.         addNode("(");
623.         pos_pointer = pos_pointer->parent;
624.         counter++;
625.         addNode(EXPRESSION);
626.         temp = expression();
627.         if(temp==-1){
628.             return -1;
629.         }
630.         pos_pointer = pos_pointer->parent;
631.         if(sym[counter].first==26){
632.             addNode(")");
633.             pos_pointer = pos_pointer->parent;
634.             counter++;
635.             return -1;
636.         }
637.     }
638.     else{
639.         return -1;
640.     }
641.
642. }
643. // 加减运算符 first={+,-}
644. int grammar_analysis::add_sub_operator(){
645.     if(sym[counter].first==14||sym[counter].first==15){
646.         counter++;
```

```
647.         return 1;
648.     }
649.     else{
650.         cout<<"Error in addition and subtract operator of "<<sym[counter].f
    irst<<endl;
651.         return -1;
652.     }
653. }
654. // 乘除运算符 first={*,/}
655. int grammar_analysis::mul_div_operator(){
656.     if(sym[counter].first==16||sym[counter].first==17){
657.         counter++;
658.         return 1;
659.     }
660.     else{
661.         cout<<"Error in multiple and division operator of "<<sym[counter].f
    irst<<endl;
662.         return -1;
663.     }
664. }
665. // 关系运算符 first={=,#,<,<=,>,>=}
666. int grammar_analysis::relation_operator(){
667.     if(sym[counter].first==19||sym[counter].first==20||sym[counter].first==
    21||sym[counter].first==22||sym[counter].first==23||sym[counter].first==24){

668.         counter++;
669.         return 1;
670.     }
671.     else{
672.         cout<<"Error in relation operator of "<<sym[counter].first<<endl;
673.         return -1;
674.     }
675. }
676. // 条件语句 first={if}
677. int grammar_analysis::conditional_statement(){
678.     int temp;
679.     if(sym[counter].first==7){
680.         addNode("if");
681.         pos_pointer = pos_pointer->parent;
682.         counter++;
683.     }
684.     else{
685.         cout<<"Error in conditional statement of"<<sym[counter].first<<endl
    ;
```

```
686.         return -1;
687.     }
688.     addNode(CONDITION);
689.     temp = condition();
690.     int jpcindex = codeRepo.size();
691.     codeRepo.push_back(code("JPC",0,-1));
692.     if(temp==-1){
693.         return -1;
694.     }
695.     pos_pointer = pos_pointer->parent;
696.     if(sym[counter].first==8){
697.         addNode("then");
698.         pos_pointer = pos_pointer->parent;
699.         counter++;
700.     }
701.     else{
702.         cout<<"Error in conditional statement of"<<sym[counter].first<<endl
    ;
703.         return -1;
704.     }
705.     addNode(STATEMENT);
706.     temp = statement();
707.     codeRepo[jpcindex].a = codeRepo.size();
708.     return temp;
709. }
710. // 过程调用语句 first={call}
711. int grammar_analysis::call_statement(){
712.     if(sym[counter].first==9){
713.         addNode("call");
714.         pos_pointer = pos_pointer->parent;
715.         counter++;
716.     }
717.     else{
718.         cout<<"Error in call statement of"<<sym[counter].first<<endl;
719.         return -1;
720.     }
721.     addNode(IDENTIFIER);
722.     int stackcount=0;
723.     int procedureIndex;//没有考虑引用没有声明的过程
724.     for(int i=0;i<table.size();i++){
725.         if(table[i].name==id[sym[counter].second]){
726.             procedureIndex = i;
727.             i++;
728.             while(i<table.size()&&table[i].kind!="PROCEDURE"){
```

```
729.                 if(table[i].kind=="VARIABLE"){
730.                     stackcount++;
731.                 }
732.                 i++;
733.             }
734.             break;
735.         }
736.     }
737.     int temp = identifier();
738.     codeRepo.push_back(code("INT",0,stackcount));
739.     codeRepo.push_back(code("CAL",table[procedureIndex].level+1-
    lev,table[procedureIndex].adr_or_value));
740.     pos_pointer = pos_pointer->parent;
741.     return temp;
742. }
743. // 当型循环语句 first={while}
744. int grammar_analysis::when_type_loop_statement(){
745.     int temp;
746.     if(sym[counter].first==10){
747.         addNode("while");
748.         pos_pointer = pos_pointer->parent;
749.         counter++;
750.     }
751.     else{
752.         cout<<"Error in while type loop statement of"<<sym[counter].first<<
    endl;
753.         return -1;
754.     }
755.     addNode(CONDITION);
756.     int jmpAddr = codeRepo.size();
757.     temp = condition();
758.     int jpcindex = codeRepo.size();
759.     codeRepo.push_back(code("JPC",0,-1));
760.     if(temp==-1){
761.         return -1;
762.     }
763.     pos_pointer = pos_pointer->parent;
764.     if(sym[counter].first==11){
765.         addNode("then");
766.         pos_pointer = pos_pointer->parent;
767.         counter++;
768.     }
769.     else{
```

```
770.         cout<<"Error in while type loop statement of"<<sym[counter].first<<
    endl;
771.         return -1;
772.     }
773.     addNode(STATEMENT);
774.     temp = statement();
775.     codeRepo.push_back(code("JMP",0,jmpAddr));
776.     codeRepo[jpcindex].a = codeRepo.size();
777.     pos_pointer = pos_pointer->parent;
778.     return temp;
779. }
780. // 读语句 first={read}
781. int grammar_analysis::read_statement(){
782.     int temp;
783.     if(sym[counter].first==12){
784.         addNode("read");
785.         pos_pointer = pos_pointer->parent;
786.         counter++;
787.     }
788.     else{
789.         cout<<"Error in read statement of"<<sym[counter].first<<endl;
790.         return -1;
791.     }
792.     if(sym[counter].first==25){
793.         addNode("(");
794.         pos_pointer = pos_pointer->parent;
795.         counter++;
796.     }
797.     else{
798.         cout<<"Error in read statement of"<<sym[counter].first<<endl;
799.         return -1;
800.     }
801.     addNode(IDENTIFIER);
802.     codeRepo.push_back(code("OPR",0,12));
803.     int tempindex = IsInTable(id[sym[counter].second]);
804.     if(tempindex==-1){
805.         cout<<id[sym[counter].second]<<"is used before statement"<<endl;
806.         return -1;
807.     }
808.     else if(table[tempindex].kind=="CONSTANT"){
809.         cout<<"Constant can not be reassigned"<<endl;
810.     }
811.     else if(table[tempindex].kind=="VARIABLE"){
```

```
812.         codeRepo.push_back(code("STO",lev-
    table[tempindex].level,table[tempindex].adr_or_value));
813.     }
814.     temp = identifier();
815.     if(temp==-1){
816.         return -1;
817.     }
818.     pos_pointer = pos_pointer->parent;
819.     while(sym[counter].first==28){
820.         addNode(",");
821.         pos_pointer = pos_pointer->parent;
822.         counter++;
823.         addNode(IDENTIFIER);
824.         codeRepo.push_back(code("OPR",0,12));
825.         int tempindex = IsInTable(id[sym[counter].second]);
826.         if(tempindex==-1){
827.             cout<<id[sym[counter].second]<<"is used before statement"<<endl
    ;
828.             return -1;
829.         }
830.         else if(table[tempindex].kind=="CONSTANT"){
831.             cout<<"Constant can not be reassigned"<<endl;
832.         }
833.         else if(table[tempindex].kind=="VARIABLE"){
834.             codeRepo.push_back(code("STO",lev-
    table[tempindex].level,table[tempindex].adr_or_value));
835.         }
836.         temp = identifier();
837.         if(temp==-1){
838.             return -1;
839.         }
840.         pos_pointer = pos_pointer->parent;
841.     }
842.     if(sym[counter].first==26){
843.         addNode(")");
844.         pos_pointer = pos_pointer->parent;
845.         counter++;
846.     }
847.     else{
848.         cout<<"Error in read statement of"<<sym[counter].first<<endl;
849.         return -1;
850.     }
851.     return 1;
852. }
```

```
853. // 写语句 first={write}
854. int grammar_analysis::write_statement(){
855.     int temp;
856.     if(sym[counter].first==13){
857.         addNode("write");
858.         pos_pointer = pos_pointer->parent;
859.         counter++;
860.     }
861.     else{
862.         cout<<"Error in write statement of"<<sym[counter].first<<endl;
863.         return -1;
864.     }
865.     if(sym[counter].first==25){
866.         addNode("(");
867.         pos_pointer = pos_pointer->parent;
868.         counter++;
869.     }
870.     else{
871.         cout<<"Error in write statement of"<<sym[counter].first<<endl;
872.         return -1;
873.     }
874.     addNode(EXPRESSION);
875.     temp = expression();
876.     codeRepo.push_back(code("OPR",0,13));
877.     if(temp==-1){
878.         return -1;
879.     }
880.     pos_pointer = pos_pointer->parent;
881.     while(sym[counter].first==28){
882.         addNode(",");
883.         pos_pointer = pos_pointer->parent;
884.         counter++;
885.         addNode(EXPRESSION);
886.         temp = expression();
887.         codeRepo.push_back(code("OPR",0,13));
888.         if(temp==-1){
889.             return -1;
890.         }
891.         pos_pointer = pos_pointer->parent;
892.     }
893.     if(sym[counter].first==26){
894.         addNode(")");
895.         pos_pointer = pos_pointer->parent;
896.         counter++;
```

```
897.     }
898.     else{
899.         cout<<"Error in write statement of"<<sym[counter].first<<endl;
900.         return -1;
901.     }
902.     return 1;
903. }
904. // 字母 first={[a-z]}
905. //int grammar_analysis::letter(){}
906. // 数字 first={[0-9]}
907. //int grammar_analysis::digit(){}
908. void grammar_analysis::drawTree(vector<node*> nodes, vector<int> nodeNum){

909.     if(nodes.size()==0){
910.         return;
911.     }
912.     for(int i=0;i<nodeNum.size();i++){
913.         cout<<nodeNum[i];
914.     }
915.     cout<<endl;
916.     vector<node*> childnodes;
917.     vector<int> childNum;
918.     int countnum = 0;
919.     int countchild = 0;
920.     for(int i=0;i<nodes.size();i++){
921.         cout<<nodes[i]->element;
922.         countchild++;
923.         while(nodeNum[countnum]==0){
924.             countnum++;
925.         }
926.         if(nodeNum[countnum]==countchild){
927.             cout<<"  ";
928.             countnum++;
929.             countchild = 0;
930.         }
931.         childNum.push_back(nodes[i]->children.size());
932.         for(int j=0;j<nodes[i]->children.size();j++){
933.             childnodes.push_back(nodes[i]->children[j]);
934.         }
935.     }
936.     cout<<endl;
937.     drawTree(childnodes,childNum);
938. }
939. void grammar_analysis::generateDot(){
```

```
940.     ofstream out("image.gv");
941.     out<<"digraph tree{"<<endl;
942.     for(int i=0;i<allNode.size();i++){
943.         out<<"element"<<allNode[i]->nodeId<<"[label="<<"\""<<allNode[i]->el
    ement<<"\"]"<<endl;
944.     }
945.     stack<node*> nodeStack;
946.     nodeStack.push(root);
947.     node* ptr;
948.     while(!nodeStack.empty()){
949.         ptr = nodeStack.top();
950.         if(ptr->parent){
951.             out<<"element"<<ptr->parent->nodeId<<" -> "<<"element"<<ptr->no
    deId<<endl;
952.         }
953.         nodeStack.pop();
954.         for(int i=0;i<ptr->children.size();i++){
955.             nodeStack.push(ptr->children[i]);
956.         }
957.     }
958.     out<<"}";
959.     out.close();
960. }
961. int grammar_analysis::convertStringToInt(const string &s){
962.     int val;
963.     strstream ss;
964.     ss << s;
965.     ss >> val;
966.     return val;
967. }
968. int grammar_analysis::IsInTable(string entity){
969.     int nextLevel = lev;
970.     for(int i=table.size()-1;i>=0;i--){
971.         if(table[i].name=="PROCEDURE"){
972.             nextLevel = table[i].level;
973.         }
974.         if(table[i].level>nextLevel){
975.             continue;
976.         }
977.         if(table[i].name == entity){
978.             return i;
979.         }
980.     }
981.     return -1;
```

```
}
void grammar_analysis::show_object_code_and_table(){
    for(int i=0;i<table.size();i++){
            cout<<i<<"   "<<table[i].name<<"    "<<table[i].kind<<"    "<<table[i].level<<"    "<<table[i].adr_or_value<<endl;
    }
    for(int i=0;i<codeRepo.size();i++){
        cout<<i<<"   "<<codeRepo[i].f<<"   "<<codeRepo[i].l<<"   "<<codeRepo[i].a<<endl;
    }
}
```

interpreter.cpp

```
#include"interpreter.h"
#include<iostream>
using namespace std;
interpreter::interpreter(vector<code> codeRepo){
    execute(codeRepo);
}
void interpreter::execute(vector<code> codeRepo){
    int I=0;//指令寄存器
    int P;//程序地址寄存器
    int T=-1;//栈顶寄存器
    int B=0;//基地址寄存器
    int S[1024];//数据栈
    for(int i=0;i<1024;i++){S[i]=0;}
    while(I<codeRepo.size()){
        //cout<<I<<"  "<<codeRepo[I].f<<"  "<<codeRepo[I].l<<"  "<<codeRepo[I].a<<"  "<<endl;
        //cout<<S[0]<<" "<<S[1]<<" "<<S[2]<<" "<<S[3]<<" "<<S[4]<<endl;
        if(codeRepo[I].f=="LIT"){
            T++;
            S[T] = codeRepo[I].a;
            I++;
        }
        else if(codeRepo[I].f=="LOD"){
            T++;
            //getBaseAddr() is aimed to get the address where the variable states.
            if(codeRepo[I].l==0){
                S[T] = S[B+codeRepo[I].a];
            }
```

```
            else{
                int index = S[B];
               // cout<<B<<"  ";
                for(int j=0;j<codeRepo[I].l-1;j++){
                    index = S[index];
                }
                S[T] = S[index+codeRepo[I].a];
                //cout<<index+codeRepo[I].a;
            }
            //cout<<"  "<<S[T]<<endl;
            I++;
        }
        else if(codeRepo[I].f=="STO"){
            if(codeRepo[I].l==0){
                S[B+codeRepo[I].a] = S[T];
            }
            else{
                int index = S[B];
                for(int j=0;j<codeRepo[I].l-1;j++){
                    index = S[index];
                }
                S[index+codeRepo[I].a] = S[T];
            }
            I++;
            T--;
        }
        else if(codeRepo[I].f=="CAL"){
            if(codeRepo[I].l==1){
                S[B] = S[B+1];
            }
            S[B+2] = I;//注意因为 opr 0 0 之后还要 I++
            //cout<<"RETURN ADDRESS  "<<B<<"  "<<S[B+2]<<"  "<<T<<endl;
            I = codeRepo[I].a;
        }
        else if(codeRepo[I].f=="INT"){
            S[T+2] = B;//DL Dynamic link
            S[T+1] = S[B];
            B = T+1;
            T += codeRepo[I].a + 3;
            //cout<<"B "<<B<<" T "<<T<<endl;
            I++;
        }
        else if(codeRepo[I].f=="JMP"){
            I = codeRepo[I].a;
```

```
        }
        else if(codeRepo[I].f=="JPC"){
            if(S[T]==0){
                I = codeRepo[I].a;
            }
            else{
                I++;
            }
        }
        else if(codeRepo[I].f=="OPR"){
            switch(codeRepo[I].a)
            {
                case 0:{
                    //cout<<B<<" "<<S[B]<<" "<<S[B+1]<<" "<<S[B+2]<<" "<<end
    l;
                    I = S[B+2];//要 cout 一下 可能有问题
                    //cout<<"I:  "<<I<<endl;
                    int tempB = S[B+1];
                    T = B-1;
                    B = tempB;
                    if(I==0){
                        return;
                    }
                    break;
                }
                case 1:{
                    S[T-1]=(S[T-1]+S[T]);
                    T--;
                    break;
                 }
                 case 2:{
                     S[T-1]=(S[T-1]-S[T]);
                     T--;
                     break;
                 }
                 case 3:{
                     S[T-1]=(S[T-1]*S[T]);
                     T--;
                     break;
                 }
                 case 4:{
                     S[T-1]=(S[T-1]/S[T]);
                     T--;
                     break;
```

```
115.                 }
116.                 case 5:{
117.                     S[T-1] = S[T-1]==S[T]?1:0;
118.                     T--;
119.                     break;
120.                 }
121.                 case 6:{
122.                     S[T-1] = S[T-1]!=S[T]?1:0;
123.                     T--;
124.                     break;
125.                 }
126.                 case 7:{
127.                     S[T-1] = S[T-1]<S[T]?1:0;
128.                     T--;
129.                     break;
130.                 }
131.                 case 8:{
132.                     S[T-1] = S[T-1]>S[T]?1:0;
133.                     T--;
134.                     break;
135.                 }
136.                 case 9:{
137.                     S[T-1] = S[T-1]<=S[T]?1:0;
138.                     T--;
139.                     break;
140.                 }
141.                 case 10:{
142.                     S[T-1] = S[T-1]>=S[T]?1:0;
143.                     T--;
144.                     break;
145.                 }
146.                 case 11:{
147.                     S[T] = S[T]%2==1?1:0;
148.                     break;
149.                 }
150.                 case 12:{
151.                     T++;
152.                     cin>>S[T];
153.                     break;
154.                 }
155.                 case 13:{
156.                     cout<<S[T]<<endl;
157.                     T--;
158.                     break;
```

```
                }
            }
            I++;
        }
    }
}
```

compiler.cpp

```
#include"grammar_analysis.h"
#include"lexical_analysis.h"
#include"interpreter.h"
#include<iostream>
using namespace std;
vector<pair<int, int> > sym;
vector<string> id;
vector<string> num;
vector<code> codeRepo;
int main(){
    lexical_analysis la(sym,id,num);
    grammar_analysis ga(sym,id,num,codeRepo);
    interpreter interpret(codeRepo);
return 0;
}
```