一、实验基础信息

> 个人信息

201700130011 —— 刘建东 —— 17级菁英班

> 实验信息

日期：2020.5.2
题目：PL/0 语言语法分析 —— 语法树生成

二、PL/0 语言文法 BNF

```
〈程序〉→〈分程序〉.
〈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(<标识符>{, <标识符>})
<字母> → a|b|c…x|y|z
<数字> → 0|1|2…7|8|9
```

> BNF 规则

符号	意义

<>	必选项
[]	可选项
{}	可重复0至无数次的项
::=	被定义为

> 关键字编号

编号	关键字	编号	关键字	编号	关键字	编号	关键字
0	.	8	begin	16	<>	24	while
1	const	9	end	17	<	25	do
2	,	10	odd	18	<=	26	read
3	;	11	+	19	>	27	write
4	=	12	-	20	>=	28	(
5	var	13	*	21	if	29	)
6	procedure	14	/	22	then	30	标识符
7	:=	15	=	23	call	31	常量

三、语法树生成过程

3.1 生成语法树结构

这部分主要是在之前语法分析的递归下降子程序中保存语法树的图结构。

首先我们需要定义一下存图方式。计算机中常见的存图方法有邻接矩阵、邻接表，空间复杂度分别是 $O(n^2)$ 和 $O(n+m)$。由于语法树的点数较多，因此选择使用 C++ 中的 $vector$ 数据结构实现邻接表存图。

```
std::vector<std::string> label;
std::vector<std::vector<int> > G;
```

在上述的存图结构中，$label$ 表示每一个节点的名称，G 存取了每一个节点的邻接节点，空间复杂度为 $O(n+m)$。

接下来我们需要考虑如何在递归下降子程序中构建出语法树。由于语法树中每一个节点只会有一个父节点，因此可以在每一个子程序中记录父节点的编号，用于将当前节点保存为父节点的

邻接点。

```
std::vector<int> empty_vec;

int Graph_init(std::string name, int fa){
    // Graph Construct
    label.push_back(name);
    G.push_back(empty_vec);
    int id = label.size()-1;
    G[fa].push_back(id);
    return id;
}
```

上述代码为生成一个名称为 "name" 的新节点，且该节点的父节点为 fa。

接下来我们可以用该函数进行语法树的构建，我们以 $Procedure()$ 和 $SubProcedure(int)$ 函数进行举例。

由于 $Procedure()$ 是整个语法分析的入口函数，因此我们不需要在该函数中传入父节点参数，即往 G 中加入了第一个叫 “程序” 的节点，并令 id 为当前节点编号，且在运行 $SubProcedure(int)$ 函数时，传入了 id 作为该函数的父节点。

```
bool Procedure() {
    // Graph Construct
    label.push_back("程序");
    G.push_back(empty_vec);
    int id = label.size()-1;

    if(!SubProcedure(id)) return Error();
    if(SYM[p1] == 0 && p1 == sz){
        Graph_init(keywords[0], id);
        return true;
    }
    else return Error();
}
```

接下来我们再看看 $Subprocedure(int)$ 函数。进入该函数后的第一件事就是创建一个新的名叫 “分程序” 的节点，并将父节点连向当前节点。之后依次识别常量、变量、过程、语句即可。

```
bool SubProcedure(int fa) {
    int id = Graph_init("分程序", fa);

    if(SYM[p1] == 1){
        if(!ConstSpecification(id)) return Error();
    }
```

```
    if(SYM[p1] == 5){
        if(!VariableSpecification(id)) return Error();
    }
    if(SYM[p1] == 6){
        if(!ProcedureSpecification(id)) return Error();
    }
    return Statement(id);
}
```

最后，我们需要将得到的图结构输出到文件，具体函数如下所示。

```
void GRAPH_OUTPUT(char *outdir) {
    std::ofstream output(outdir);
    int n = label.size();
    output << n << std::endl;
    for(int i = 0; i < n; i++)
        output << label[i] << " \n"[i==n-1];
    for(int i = 0; i < n; i++){
        int sz = G[i].size();
        if(sz == 0) output << std::endl;
        for(int j = 0; j < sz; j++)
            output << G[i][j] << " \n"[j==sz-1];
    }
    output.close();
}
```

3.2 画出语法树

由于 C++ 画图比较麻烦，因此我们使用 $python$ 中的 $graphviz$ 库来画出语法树。

画出语法树主要分为两步：

1. 从文件中读出之前生成的树结构
2. dfs 遍历语法树

首先是第一步，读出树结构，其中 n 表示树中节点个数，$lable$ 即为每个节点的名称，G 存储图结构。

```
def data_acquire(path):
    # read file
    fo = open(path, "r+")
    data = fo.read().split("\n")

    # acquire data
    n = int(data[0])
    lable = data[1].split(" ")
```

```
    G = []
    for row in range(2, len(data)):
        G.append(data[row].split(" "))
    return n, lable, G
```

接下来是第二步，dfs 遍历图，其中 x 为当前节点，fa 为父节点。

```
def dfs(x, fa, lable, G, graph):
    graph.node(str(x), lable[x])
    if fa != -1:
        graph.edge(str(fa), str(x))
    for y in G[x]:
        if y == "":
            break
        dfs(int(y), x, lable, G, graph)
```

最后我们可以调用库函数得到构建的语法树以及树内节点具体连边信息。

四、运行结果

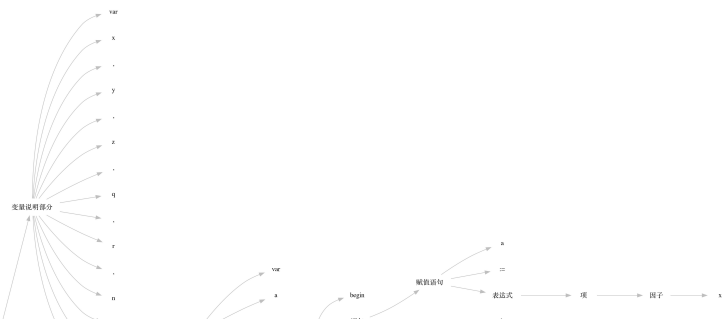
此部分我们将给出一份 PL/0 文法的程序及其对应的语法树结构、语法树图。

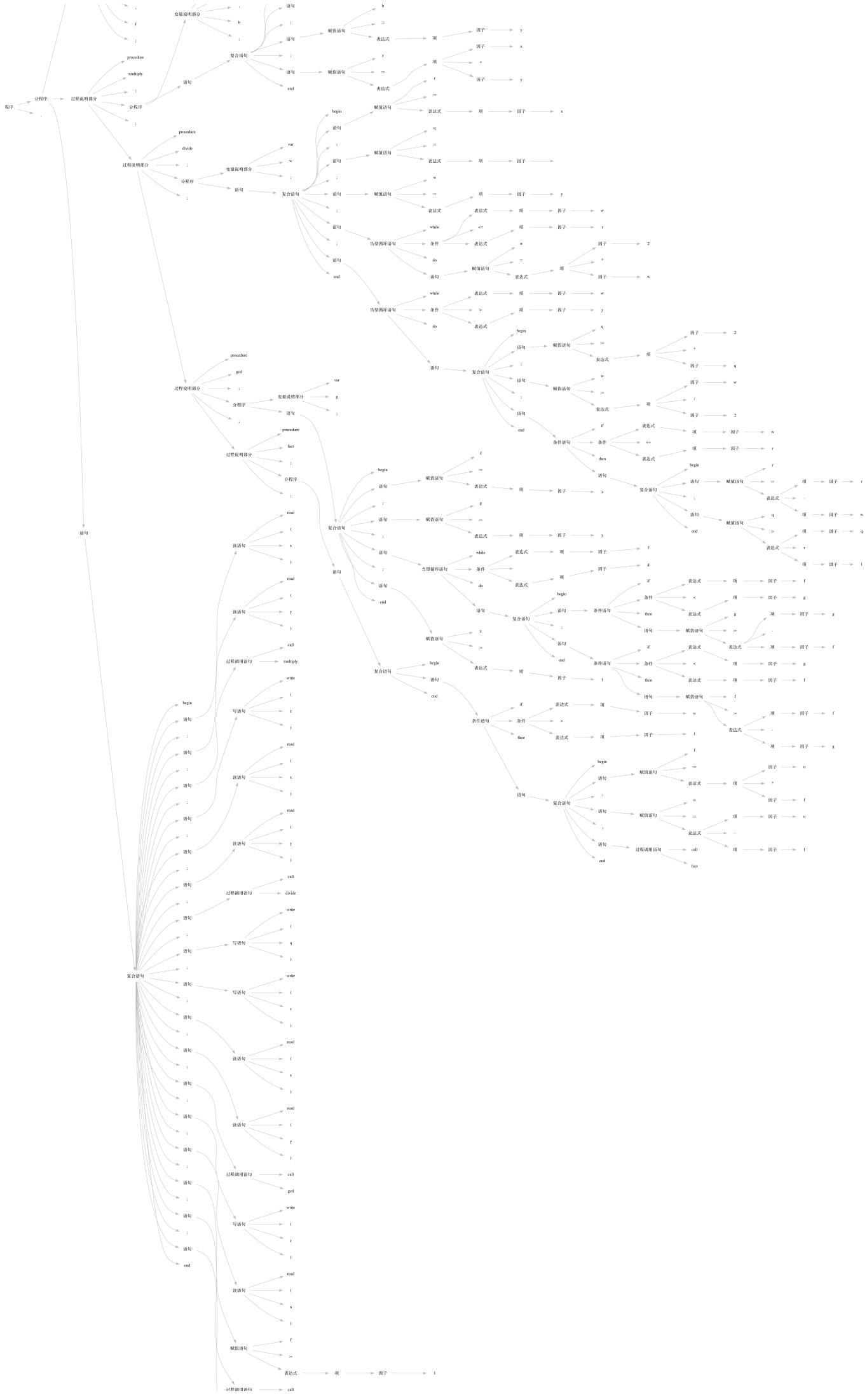
4.1 PL/0 文法程序

```
var x, y, z, q, r, n, f;

procedure multiply;
var a, b;
begin
  a := x;
  b := y;
  z := x*y
end;

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.
```

4.2 语法树图




程序
分程序
.
过程说明部分
procedure
multiply
;
分程序
;
变量说明部分
,
b
;
语句
复合语句
语句
;
语句
;
语句
end
赋值语句
b
:=
表达式
项
因子
y
因子
x
*
因子
y
赋值语句
z
:=
表达式
项
r
:=
表达式
项
因子
x
过程说明部分
procedure
divide
;
分程序
;
变量说明部分
var
w
;
语句
复合语句
begin
语句
;
语句
;
语句
;
语句
;
语句
end
赋值语句
q
:=
表达式
项
因子
赋值语句
w
:=
表达式
项
因子
y
当型循环语句
while
条件
表达式
<=
表达式
项
因子
w
项
因子
r
do
语句
赋值语句
w
:=
表达式
项
因子
2
*
因子
w
当型循环语句
while
条件
表达式
项
因子
w
>
表达式
项
因子
y
do
语句
复合语句
begin
语句
赋值语句
q
:=
表达式
项
因子
2
*
因子
q
;
语句
赋值语句
w
:=
表达式
项
因子
w
/
因子
2
;
语句
end
条件语句
if
条件
表达式
项
因子
w
<=
表达式
项
因子
r
then
语句
复合语句
begin
语句
赋值语句
r
:=
表达式
项
因子
r
-
项
因子
w
;
语句
赋值语句
q
:=
表达式
项
因子
q
+
项
因子
l
end
过程说明部分
procedure
gcd
;
分程序
;
变量说明部分
var
g
;
语句
复合语句
begin
语句
赋值语句
f
:=
表达式
项
因子
x
;
语句
赋值语句
g
:=
表达式
项
因子
y
;
语句
当型循环语句
while
条件
表达式
项
因子
f
do
语句
复合语句
begin
语句
条件语句
if
条件
表达式
项
因子
f
<
表达式
项
因子
g
then
语句
赋值语句
g
:=
表达式
项
因子
g
-
项
因子
f
;
语句
条件语句
if
条件
表达式
项
因子
f
<
表达式
项
因子
g
then
语句
赋值语句
f
:=
表达式
项
因子
f
-
项
因子
g
end
赋值语句
z
:=
表达式
项
因子
f
;
语句
end
条件语句
if
条件
表达式
项
因子
n
>
表达式
项
因子
l
then
语句
复合语句
begin
语句
赋值语句
f
:=
表达式
项
因子
n
*
因子
f
;
语句
赋值语句
n
:=
表达式
项
因子
n
-
项
因子
l
;
语句
过程调用语句
call
fact
end
;
语句
end
语句
复合语句
begin
语句
流语句
read
(
x
)
;
语句
流语句
read
(
y
)
;
语句
过程调用语句
call
multiply
;
语句
写语句
write
(
z
)
;
语句
流语句
read
(
x
)
;
语句
流语句
read
(
y
)
;
语句
过程调用语句
call
divide
;
语句
写语句
write
(
q
)
;
语句
写语句
write
(
r
)
;
语句
流语句
read
(
x
)
;
语句
流语句
read
(
y
)
;
语句
过程调用语句
call
gcd
;
语句
写语句
write
(
z
)
;
语句
流语句
read
(
n
)
;
语句
赋值语句
f
:=
表达式
项
因子
1
;
语句
过程调用语句
call

4.3 语法树结构

```
digraph G {
    node [color=white style=filled]
    edge [color=grey style=filled]
    bgcolor=white rankdir=LR style=filled
    0 [label="程序"]
    1 [label="分程序"]
    0 -> 1
    2 [label="变量说明部分"]
    1 -> 2
    3 [label=var]
    2 -> 3
    4 [label=x]
    2 -> 4
    5 [label=","]
    2 -> 5
    6 [label=y]
    2 -> 6
    7 [label=","]
    2 -> 7
    8 [label=z]
    2 -> 8
    9 [label=","]
    2 -> 9
    10 [label=q]
    2 -> 10
    11 [label=","]
    2 -> 11
    12 [label=r]
    2 -> 12
    13 [label=","]
    2 -> 13
    14 [label=n]
    2 -> 14
    15 [label=","]
    2 -> 15
    16 [label=f]
    2 -> 16
    17 [label=";"]
    2 -> 17
    18 [label="过程说明部分"]
```

```
1 -> 18
19 [label=procedure]
18 -> 19
20 [label=multiply]
18 -> 20
21 [label=";"]
18 -> 21
22 [label="分程序"]
18 -> 22
23 [label="变量说明部分"]
22 -> 23
24 [label=var]
23 -> 24
25 [label=a]
23 -> 25
26 [label=","]
23 -> 26
27 [label=b]
23 -> 27
28 [label=";"]
23 -> 28
29 [label="语句"]
22 -> 29
30 [label="复合语句"]
29 -> 30
31 [label=begin]
30 -> 31
32 [label="语句"]
30 -> 32
33 [label="赋值语句"]
32 -> 33
34 [label=a]
33 -> 34
35 [label=":="]
33 -> 35
36 [label="表达式"]
33 -> 36
37 [label="项"]
36 -> 37
38 [label="因子"]
37 -> 38
39 [label=x]
38 -> 39
40 [label=";"]
30 -> 40
41 [label="语句"]
30 -> 41
42 [label="赋值语句"]
```

```
41 -> 42
43 [label=b]
42 -> 43
44 [label=":="]
42 -> 44
45 [label="表达式"]
42 -> 45
46 [label="项"]
45 -> 46
47 [label="因子"]
46 -> 47
48 [label=y]
47 -> 48
49 [label=";"]
30 -> 49
50 [label="语句"]
30 -> 50
51 [label="赋值语句"]
50 -> 51
52 [label=z]
51 -> 52
53 [label=":="]
51 -> 53
54 [label="表达式"]
51 -> 54
55 [label="项"]
54 -> 55
56 [label="因子"]
55 -> 56
57 [label=x]
56 -> 57
58 [label="*"]
55 -> 58
59 [label="因子"]
55 -> 59
60 [label=y]
59 -> 60
61 [label=end]
30 -> 61
62 [label=";"]
18 -> 62
63 [label="过程说明部分"]
18 -> 63
64 [label=procedure]
63 -> 64
65 [label=divide]
63 -> 65
66 [label=";"]
```

```
63 -> 66
67 [label="分程序"]
63 -> 67
68 [label="变量说明部分"]
67 -> 68
69 [label=var]
68 -> 69
70 [label=w]
68 -> 70
71 [label=";"]
68 -> 71
72 [label="语句"]
67 -> 72
73 [label="复合语句"]
72 -> 73
74 [label=begin]
73 -> 74
75 [label="语句"]
73 -> 75
76 [label="赋值语句"]
75 -> 76
77 [label=r]
76 -> 77
78 [label=":="]
76 -> 78
79 [label="表达式"]
76 -> 79
80 [label="项"]
79 -> 80
81 [label="因子"]
80 -> 81
82 [label=x]
81 -> 82
83 [label=";"]
73 -> 83
84 [label="语句"]
73 -> 84
85 [label="赋值语句"]
84 -> 85
86 [label=q]
85 -> 86
87 [label=":="]
85 -> 87
88 [label="表达式"]
85 -> 88
89 [label="项"]
88 -> 89
90 [label="因子"]
```

```
89 -> 90
91 [label=""]
90 -> 91
92 [label=";"]
73 -> 92
93 [label="语句"]
73 -> 93
94 [label="赋值语句"]
93 -> 94
95 [label=w]
94 -> 95
96 [label=":="]
94 -> 96
97 [label="表达式"]
94 -> 97
98 [label="项"]
97 -> 98
99 [label="因子"]
98 -> 99
100 [label=y]
99 -> 100
101 [label=";"]
73 -> 101
102 [label="语句"]
73 -> 102
103 [label="当型循环语句"]
102 -> 103
104 [label=while]
103 -> 104
105 [label="条件"]
103 -> 105
106 [label="表达式"]
105 -> 106
107 [label="项"]
106 -> 107
108 [label="因子"]
107 -> 108
109 [label=w]
108 -> 109
110 [label="<="]
105 -> 110
111 [label="表达式"]
105 -> 111
112 [label="项"]
111 -> 112
113 [label="因子"]
112 -> 113
114 [label=r]
```

```
113 -> 114
115 [label=do]
103 -> 115
116 [label="语句"]
103 -> 116
117 [label="赋值语句"]
116 -> 117
118 [label=w]
117 -> 118
119 [label=":="]
117 -> 119
120 [label="表达式"]
117 -> 120
121 [label="项"]
120 -> 121
122 [label="因子"]
121 -> 122
123 [label=2]
122 -> 123
124 [label="*"]
121 -> 124
125 [label="因子"]
121 -> 125
126 [label=w]
125 -> 126
127 [label=";"]
73 -> 127
128 [label="语句"]
73 -> 128
129 [label="当型循环语句"]
128 -> 129
130 [label=while]
129 -> 130
131 [label="条件"]
129 -> 131
132 [label="表达式"]
131 -> 132
133 [label="项"]
132 -> 133
134 [label="因子"]
133 -> 134
135 [label=w]
134 -> 135
136 [label=">"]
131 -> 136
137 [label="表达式"]
131 -> 137
138 [label="项"]
```

```
137 -> 138
139 [label="因子"]
138 -> 139
140 [label=y]
139 -> 140
141 [label=do]
129 -> 141
142 [label="语句"]
129 -> 142
143 [label="复合语句"]
142 -> 143
144 [label=begin]
143 -> 144
145 [label="语句"]
143 -> 145
146 [label="赋值语句"]
145 -> 146
147 [label=q]
146 -> 147
148 [label=":="]
146 -> 148
149 [label="表达式"]
146 -> 149
150 [label="项"]
149 -> 150
151 [label="因子"]
150 -> 151
152 [label=2]
151 -> 152
153 [label="*"]
150 -> 153
154 [label="因子"]
150 -> 154
155 [label=q]
154 -> 155
156 [label=";"]
143 -> 156
157 [label="语句"]
143 -> 157
158 [label="赋值语句"]
157 -> 158
159 [label=w]
158 -> 159
160 [label=":="]
158 -> 160
161 [label="表达式"]
158 -> 161
162 [label="项"]
```

```
161 -> 162
163 [label="因子"]
162 -> 163
164 [label=w]
163 -> 164
165 [label="/"]
162 -> 165
166 [label="因子"]
162 -> 166
167 [label=2]
166 -> 167
168 [label=";"]
143 -> 168
169 [label="语句"]
143 -> 169
170 [label="条件语句"]
169 -> 170
171 [label=if]
170 -> 171
172 [label="条件"]
170 -> 172
173 [label="表达式"]
172 -> 173
174 [label="项"]
173 -> 174
175 [label="因子"]
174 -> 175
176 [label=w]
175 -> 176
177 [label="<="]
172 -> 177
178 [label="表达式"]
172 -> 178
179 [label="项"]
178 -> 179
180 [label="因子"]
179 -> 180
181 [label=r]
180 -> 181
182 [label=then]
170 -> 182
183 [label="语句"]
170 -> 183
184 [label="复合语句"]
183 -> 184
185 [label=begin]
184 -> 185
186 [label="语句"]
```

```
184 -> 186
187 [label="赋值语句"]
186 -> 187
188 [label=r]
187 -> 188
189 [label=":="]
187 -> 189
190 [label="表达式"]
187 -> 190
191 [label="项"]
190 -> 191
192 [label="因子"]
191 -> 192
193 [label=r]
192 -> 193
194 [label="-"]
190 -> 194
195 [label="项"]
190 -> 195
196 [label="因子"]
195 -> 196
197 [label=w]
196 -> 197
198 [label=";"]
184 -> 198
199 [label="语句"]
184 -> 199
200 [label="赋值语句"]
199 -> 200
201 [label=q]
200 -> 201
202 [label=":="]
200 -> 202
203 [label="表达式"]
200 -> 203
204 [label="项"]
203 -> 204
205 [label="因子"]
204 -> 205
206 [label=q]
205 -> 206
207 [label="+"]
203 -> 207
208 [label="项"]
203 -> 208
209 [label="因子"]
208 -> 209
210 [label=1]
```

```
209 -> 210
211 [label=end]
184 -> 211
212 [label=end]
143 -> 212
213 [label=end]
73 -> 213
214 [label=";"]
63 -> 214
215 [label="过程说明部分"]
63 -> 215
216 [label=procedure]
215 -> 216
217 [label=gcd]
215 -> 217
218 [label=";"]
215 -> 218
219 [label="分程序"]
215 -> 219
220 [label="变量说明部分"]
219 -> 220
221 [label=var]
220 -> 221
222 [label=g]
220 -> 222
223 [label=";"]
220 -> 223
224 [label="语句"]
219 -> 224
225 [label="复合语句"]
224 -> 225
226 [label=begin]
225 -> 226
227 [label="语句"]
225 -> 227
228 [label="赋值语句"]
227 -> 228
229 [label=f]
228 -> 229
230 [label=":="]
228 -> 230
231 [label="表达式"]
228 -> 231
232 [label="项"]
231 -> 232
233 [label="因子"]
232 -> 233
234 [label=x]
```

```
233 -> 234
235 [label=";"]
225 -> 235
236 [label="语句"]
225 -> 236
237 [label="赋值语句"]
236 -> 237
238 [label=g]
237 -> 238
239 [label=":="]
237 -> 239
240 [label="表达式"]
237 -> 240
241 [label="项"]
240 -> 241
242 [label="因子"]
241 -> 242
243 [label=y]
242 -> 243
244 [label=";"]
225 -> 244
245 [label="语句"]
225 -> 245
246 [label="当型循环语句"]
245 -> 246
247 [label=while]
246 -> 247
248 [label="条件"]
246 -> 248
249 [label="表达式"]
248 -> 249
250 [label="项"]
249 -> 250
251 [label="因子"]
250 -> 251
252 [label=f]
251 -> 252
253 [label=<>]
248 -> 253
254 [label="表达式"]
248 -> 254
255 [label="项"]
254 -> 255
256 [label="因子"]
255 -> 256
257 [label=g]
256 -> 257
258 [label=do]
```

```
246 -> 258
259 [label="语句"]
246 -> 259
260 [label="复合语句"]
259 -> 260
261 [label=begin]
260 -> 261
262 [label="语句"]
260 -> 262
263 [label="条件语句"]
262 -> 263
264 [label=if]
263 -> 264
265 [label="条件"]
263 -> 265
266 [label="表达式"]
265 -> 266
267 [label="项"]
266 -> 267
268 [label="因子"]
267 -> 268
269 [label=f]
268 -> 269
270 [label="<"]
265 -> 270
271 [label="表达式"]
265 -> 271
272 [label="项"]
271 -> 272
273 [label="因子"]
272 -> 273
274 [label=g]
273 -> 274
275 [label=then]
263 -> 275
276 [label="语句"]
263 -> 276
277 [label="赋值语句"]
276 -> 277
278 [label=g]
277 -> 278
279 [label=":="]
277 -> 279
280 [label="表达式"]
277 -> 280
281 [label="项"]
280 -> 281
282 [label="因子"]
```

```
281 -> 282
283 [label=g]
282 -> 283
284 [label="-"]
280 -> 284
285 [label="项"]
280 -> 285
286 [label="因子"]
285 -> 286
287 [label=f]
286 -> 287
288 [label=";"]
260 -> 288
289 [label="语句"]
260 -> 289
290 [label="条件语句"]
289 -> 290
291 [label=if]
290 -> 291
292 [label="条件"]
290 -> 292
293 [label="表达式"]
292 -> 293
294 [label="项"]
293 -> 294
295 [label="因子"]
294 -> 295
296 [label=g]
295 -> 296
297 [label="<"]
292 -> 297
298 [label="表达式"]
292 -> 298
299 [label="项"]
298 -> 299
300 [label="因子"]
299 -> 300
301 [label=f]
300 -> 301
302 [label=then]
290 -> 302
303 [label="语句"]
290 -> 303
304 [label="赋值语句"]
303 -> 304
305 [label=f]
304 -> 305
306 [label=":="]
```

```
304 -> 306
307 [label="表达式"]
304 -> 307
308 [label="项"]
307 -> 308
309 [label="因子"]
308 -> 309
310 [label=f]
309 -> 310
311 [label="-"]
307 -> 311
312 [label="项"]
307 -> 312
313 [label="因子"]
312 -> 313
314 [label=g]
313 -> 314
315 [label=end]
260 -> 315
316 [label=";"]
225 -> 316
317 [label="语句"]
225 -> 317
318 [label="赋值语句"]
317 -> 318
319 [label=z]
318 -> 319
320 [label=":="]
318 -> 320
321 [label="表达式"]
318 -> 321
322 [label="项"]
321 -> 322
323 [label="因子"]
322 -> 323
324 [label=f]
323 -> 324
325 [label=end]
225 -> 325
326 [label=";"]
215 -> 326
327 [label="过程说明部分"]
215 -> 327
328 [label=procedure]
327 -> 328
329 [label=fact]
327 -> 329
330 [label=";"]
```

```
327 -> 330
331 [label="分程序"]
327 -> 331
332 [label="语句"]
331 -> 332
333 [label="复合语句"]
332 -> 333
334 [label=begin]
333 -> 334
335 [label="语句"]
333 -> 335
336 [label="条件语句"]
335 -> 336
337 [label=if]
336 -> 337
338 [label="条件"]
336 -> 338
339 [label="表达式"]
338 -> 339
340 [label="项"]
339 -> 340
341 [label="因子"]
340 -> 341
342 [label=n]
341 -> 342
343 [label=">"]
338 -> 343
344 [label="表达式"]
338 -> 344
345 [label="项"]
344 -> 345
346 [label="因子"]
345 -> 346
347 [label=1]
346 -> 347
348 [label=then]
336 -> 348
349 [label="语句"]
336 -> 349
350 [label="复合语句"]
349 -> 350
351 [label=begin]
350 -> 351
352 [label="语句"]
350 -> 352
353 [label="赋值语句"]
352 -> 353
354 [label=f]
```

```
353 -> 354
355 [label=":="]
353 -> 355
356 [label="表达式"]
353 -> 356
357 [label="项"]
356 -> 357
358 [label="因子"]
357 -> 358
359 [label=n]
358 -> 359
360 [label="*"]
357 -> 360
361 [label="因子"]
357 -> 361
362 [label=f]
361 -> 362
363 [label=";"]
350 -> 363
364 [label="语句"]
350 -> 364
365 [label="赋值语句"]
364 -> 365
366 [label=n]
365 -> 366
367 [label=":="]
365 -> 367
368 [label="表达式"]
365 -> 368
369 [label="项"]
368 -> 369
370 [label="因子"]
369 -> 370
371 [label=n]
370 -> 371
372 [label="-"]
368 -> 372
373 [label="项"]
368 -> 373
374 [label="因子"]
373 -> 374
375 [label=1]
374 -> 375
376 [label=";"]
350 -> 376
377 [label="语句"]
350 -> 377
378 [label="过程调用语句"]
```

```
377 -> 378
379 [label=call]
378 -> 379
380 [label=fact]
378 -> 380
381 [label=end]
350 -> 381
382 [label=end]
333 -> 382
383 [label=";"]
327 -> 383
384 [label="语句"]
1 -> 384
385 [label="复合语句"]
384 -> 385
386 [label=begin]
385 -> 386
387 [label="语句"]
385 -> 387
388 [label="读语句"]
387 -> 388
389 [label=read]
388 -> 389
390 [label="("]
388 -> 390
391 [label=x]
388 -> 391
392 [label=")"]
388 -> 392
393 [label=";"]
385 -> 393
394 [label="语句"]
385 -> 394
395 [label="读语句"]
394 -> 395
396 [label=read]
395 -> 396
397 [label="("]
395 -> 397
398 [label=y]
395 -> 398
399 [label=")"]
395 -> 399
400 [label=";"]
385 -> 400
401 [label="语句"]
385 -> 401
402 [label="过程调用语句"]
```

```
401 -> 402
403 [label=call]
402 -> 403
404 [label=multiply]
402 -> 404
405 [label=";"]
385 -> 405
406 [label="语句"]
385 -> 406
407 [label="写语句"]
406 -> 407
408 [label=write]
407 -> 408
409 [label="("]
407 -> 409
410 [label=z]
407 -> 410
411 [label=")"]
407 -> 411
412 [label=";"]
385 -> 412
413 [label="语句"]
385 -> 413
414 [label="读语句"]
413 -> 414
415 [label=read]
414 -> 415
416 [label="("]
414 -> 416
417 [label=x]
414 -> 417
418 [label=")"]
414 -> 418
419 [label=";"]
385 -> 419
420 [label="语句"]
385 -> 420
421 [label="读语句"]
420 -> 421
422 [label=read]
421 -> 422
423 [label="("]
421 -> 423
424 [label=y]
421 -> 424
425 [label=")"]
421 -> 425
426 [label=";"]
```

```
385 -> 426
427 [label="语句"]
385 -> 427
428 [label="过程调用语句"]
427 -> 428
429 [label=call]
428 -> 429
430 [label=divide]
428 -> 430
431 [label=";"]
385 -> 431
432 [label="语句"]
385 -> 432
433 [label="写语句"]
432 -> 433
434 [label=write]
433 -> 434
435 [label="("]
433 -> 435
436 [label=q]
433 -> 436
437 [label=")"]
433 -> 437
438 [label=";"]
385 -> 438
439 [label="语句"]
385 -> 439
440 [label="写语句"]
439 -> 440
441 [label=write]
440 -> 441
442 [label="("]
440 -> 442
443 [label=r]
440 -> 443
444 [label=")"]
440 -> 444
445 [label=";"]
385 -> 445
446 [label="语句"]
385 -> 446
447 [label="读语句"]
446 -> 447
448 [label=read]
447 -> 448
449 [label="("]
447 -> 449
450 [label=x]
```

```
447 -> 450
451 [label=")"]
447 -> 451
452 [label=";"]
385 -> 452
453 [label="语句"]
385 -> 453
454 [label="读语句"]
453 -> 454
455 [label=read]
454 -> 455
456 [label="("]
454 -> 456
457 [label=y]
454 -> 457
458 [label=")"]
454 -> 458
459 [label=";"]
385 -> 459
460 [label="语句"]
385 -> 460
461 [label="过程调用语句"]
460 -> 461
462 [label=call]
461 -> 462
463 [label=gcd]
461 -> 463
464 [label=";"]
385 -> 464
465 [label="语句"]
385 -> 465
466 [label="写语句"]
465 -> 466
467 [label=write]
466 -> 467
468 [label="("]
466 -> 468
469 [label=z]
466 -> 469
470 [label=")"]
466 -> 470
471 [label=";"]
385 -> 471
472 [label="语句"]
385 -> 472
473 [label="读语句"]
472 -> 473
474 [label=read]
```

```
473 -> 474
475 [label="("]
473 -> 475
476 [label=n]
473 -> 476
477 [label=")"]
473 -> 477
478 [label=";"]
385 -> 478
479 [label="语句"]
385 -> 479
480 [label="赋值语句"]
479 -> 480
481 [label=f]
480 -> 481
482 [label=":="]
480 -> 482
483 [label="表达式"]
480 -> 483
484 [label="项"]
483 -> 484
485 [label="因子"]
484 -> 485
486 [label=1]
485 -> 486
487 [label=";"]
385 -> 487
488 [label="语句"]
385 -> 488
489 [label="过程调用语句"]
488 -> 489
490 [label=call]
489 -> 490
491 [label=fact]
489 -> 491
492 [label=";"]
385 -> 492
493 [label="语句"]
385 -> 493
494 [label="写语句"]
493 -> 494
495 [label=write]
494 -> 495
496 [label="("]
494 -> 496
497 [label=f]
494 -> 497
498 [label=")"]
```

```
    494 -> 498
    499 [label=end]
    385 -> 499
    500 [label="."]
    0 -> 500
}
```

五、源代码

5.1 syntax_analyzer.cpp

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <algorithm>
#include "lexical_analyzer.h"
#include "syntax_analyzer.h"

const std::string CONSTANT = "CONSTANT", VARIABLE = "VARIABLE", PROCEDURE = "PROC
EDURE";
const int DX = 3;
int TX = 0, LEV = 0, ADR = DX, p1, p2, p3, sz;

std::vector<Table> TABLE;
std::vector<int> empty_vec;
std::vector<std::string> label;
std::vector<std::vector<int> > G;

void DFS_OUTPUT(int TableNum, int EntryNum, int flag, std::ofstream &output) {
    // Table Header Output
    if(flag == 1){
        output << std::left << std::setfill('-') << std::setw(100) << "-" << std:
:endl;
        output << "TX: " << std::left << std::setfill(' ') << std::setw(10) << Ta
bleNum;
        if(EntryNum >= TABLE[TableNum].table.size()){
            output << std::endl;
            return;
        }
        flag = 0;
    }
    else output << "     " << std::left << std::setfill(' ') << std::setw(10) << "
 ";

    // Table Information Output
```

```
    output << "NAME: " << std::left << std::setw(20) << TABLE[TableNum].table[Ent
ryNum].NAME;
    output << "KIND: " << std::left << std::setw(25) << TABLE[TableNum].table[Ent
ryNum].KIND;
    if(TABLE[TableNum].table[EntryNum].KIND == CONSTANT)
        output << "VAL: " << std::left << std::setw(15) << TABLE[TableNum].table[
EntryNum].VAL << std::endl;
    else
        output << "LEVEL: " << std::left << std::setw(15) << TABLE[TableNum].tabl
e[EntryNum].LEVEL
                  << "ADR: " << TABLE[TableNum].table[EntryNum].ADR << std::endl;

    // Recursion & Backtrack Output
    if(TABLE[TableNum].table[EntryNum].KIND == PROCEDURE) DFS_OUTPUT(TABLE[TableN
um].table[EntryNum].NXT, 0, 1, output), flag = 1;
    if(EntryNum+1 < TABLE[TableNum].table.size()) DFS_OUTPUT(TableNum, EntryNum+1
, flag, output);
}

void TABLE_OUTPUT(char *outdir) {
    std::ofstream output(outdir);
    DFS_OUTPUT(0, 0, 1, output);
    output << std::left << std::setfill('-') << std::setw(100) << "-" << std::end
l;
    output.close();
}

void GRAPH_OUTPUT(char *outdir) {
    std::ofstream output(outdir);
    int n = label.size();
    output << n << std::endl;
    for(int i = 0; i < n; i++)
        output << label[i] << " \n"[i==n-1];
    for(int i = 0; i < n; i++){
        int sz = G[i].size();
        if(sz == 0) output << std::endl;
        for(int j = 0; j < sz; j++)
            output << G[i][j] << " \n"[j==sz-1];
    }
    output.close();
}

bool Error() {
    std::cout << "Error! p1: " << p1 << std::endl;
    return false;
}

std::string ll2string(long long x){
```

```
    std::string result = "";
    while(x){
        result += '0'+(x%10);
        x /= 10;
    }
    reverse(result.begin(), result.end());
    return result;
}

int Graph_init(std::string name, int fa){
    // Graph Construct
    label.push_back(name);
    G.push_back(empty_vec);
    int id = label.size()-1;
    G[fa].push_back(id);
    return id;
}

bool BLOCK() {
    TABLE.push_back(Table());
    p1 = p2 = p3 = 0; sz = SYM.size()-1;
    label.clear(); G.clear();
    return Procedure();
}

/*------------------- Procedure Function -------------------*/
bool Procedure() {
    // Graph Construct
    label.push_back("程序");
    G.push_back(empty_vec);
    int id = label.size()-1;

    if(!SubProcedure(id)) return Error();
    if(SYM[p1] == 0 && p1 == sz){
        Graph_init(keywords[0], id);
        return true;
    }
    else return Error();
}

bool SubProcedure(int fa) {
    int id = Graph_init("分程序", fa);

    if(SYM[p1] == 1){
        if(!ConstSpecification(id)) return Error();
    }
    if(SYM[p1] == 5){
        if(!VariableSpecification(id)) return Error();
```

```
    }
    if(SYM[p1] == 6){
        if(!ProcedureSpecification(id)) return Error();
    }
    return Statement(id);
}

/*------------------- Specification Function -------------------*/
bool ConstDefinition(int fa) {
    int id = Graph_init("常量定义", fa);
    TableEntry entry;
    if(SYM[p1] == tagtype) {
        entry.NAME = ID[p2++];
        entry.KIND = CONSTANT;
        p1++;
        Graph_init(entry.NAME, id);
    }
    else return Error();

    Judge(4);
    Graph_init(keywords[4], id);

    if(SYM[p1] == numtype) {
        entry.VAL = NUM[p3++];
        entry.LEVEL = LEV;
        TABLE[TX].table.push_back(entry);
        p1++;
        Graph_init(ll2string(entry.VAL), id);
        return true;
    }
    else return Error();
}

bool ConstSpecification(int fa) {
    int id = Graph_init("常量说明部分", fa);
    Judge(1);
    Graph_init(keywords[1], id);

    if(!ConstDefinition(id)) return Error();

    while(SYM[p1] == 2){
        p1++;
        Graph_init(keywords[2], id);
        if(!ConstDefinition(id)) return Error();
    }
    Judge(3);
    Graph_init(keywords[3], id);
    return true;
```

```
}

void RecognizeFlag(std::string flag, int fa) {
    TableEntry entry;
    entry.NAME = ID[p2++];
    entry.KIND = flag;
    entry.LEVEL = LEV;
    entry.ADR = ADR++;
    TABLE[TX].table.push_back(entry);
    p1++;
    Graph_init(entry.NAME, fa);
}

bool VariableSpecification(int fa) {
    int id = Graph_init("变量说明部分", fa);
    Judge(5);
    Graph_init(keywords[5], id);
    if(SYM[p1] == tagtype) RecognizeFlag(VARIABLE, id);
    else return Error();

    while(SYM[p1] == 2) {
        p1++;
        Graph_init(keywords[2], id);
        if(SYM[p1] == tagtype) RecognizeFlag(VARIABLE, id);
        else return Error();
    }
    Judge(3);
    Graph_init(keywords[3], id);
    return true;
}

bool ProcedureSpecification(int fa) {
    int id = Graph_init("过程说明部分", fa);
    Judge(6);
    Graph_init(keywords[6], id);
    if(SYM[p1] == tagtype){
        RecognizeFlag(PROCEDURE, id);
        TABLE.push_back(Table());
        TABLE[TABLE.size()-1].PreTable = TX;
        TABLE[TABLE.size()-1].PreEntry = TABLE[TX].table.size()-1;
        TABLE[TX].table[TABLE[TX].table.size()-1].NXT = TABLE.size()-1;
        TX = TABLE.size()-1, LEV++, ADR = DX;
    }
    else return Error();

    Judge(3);
    Graph_init(keywords[3], id);
    if(!SubProcedure(id)) return Error();
```

```
    Judge(3);
    Graph_init(keywords[3], id);
    LEV--;
    int PreEntry = TABLE[TX].PreEntry;
    TX = TABLE[TX].PreTable;
    ADR = ADR - 1 + TABLE[TX].table[PreEntry].ADR;
    TABLE[TX].table[PreEntry].ADR = ADR++;

    while(SYM[p1] == 6){
        if(!ProcedureSpecification(id)) return Error();
    }
    return true;
}

/*------------------- Statement Function -------------------*/
bool Statement(int fa) {
    if(SYM[p1] == tagtype) return AssignStatement(Graph_init("语句", fa));
    if(SYM[p1] == 21) return ConditionStatement(Graph_init("语句", fa));
    if(SYM[p1] == 24) return LoopStatement(Graph_init("语句", fa));
    if(SYM[p1] == 23) return CallStatement(Graph_init("语句", fa));
    if(SYM[p1] == 26) return ReadStatement(Graph_init("语句", fa));
    if(SYM[p1] == 27) return WriteStatement(Graph_init("语句", fa));
    if(SYM[p1] == 8) return CompoundStatement(Graph_init("语句", fa));
    return true;
}

bool AssignStatement(int fa) {
    int id = Graph_init("赋值语句", fa);
    if(SYM[p1] == tagtype) Graph_init(ID[p2++], id), p1++;
    else return Error();

    Judge(7);
    Graph_init(keywords[7], id);
    return Expression(id);
}

bool ConditionStatement(int fa) {
    int id = Graph_init("条件语句", fa);
    Judge(21);
    Graph_init(keywords[21], id);
    if(!Condition(id)) return Error();
    Judge(22);
    Graph_init(keywords[22], id);
    return Statement(id);
}

bool LoopStatement(int fa) {
    int id = Graph_init("当型循环语句", fa);
```

```
    Judge(24);
    Graph_init(keywords[24], id);
    if(!Condition(id)) return Error();
    Judge(25);
    Graph_init(keywords[25], id);
    return Statement(id);
}

bool CallStatement(int fa) {
    int id = Graph_init("过程调用语句", fa);
    Judge(23);
    Graph_init(keywords[23], id);
    if(SYM[p1] == tagtype) Graph_init(ID[p2++], id), p1++;
    else return Error();
    return true;
}

bool ReadStatement(int fa) {
    int id = Graph_init("读语句", fa);
    Judge(26);
    Graph_init(keywords[26], id);
    Judge(28);
    Graph_init(keywords[28], id);
    if(SYM[p1] == tagtype) Graph_init(ID[p2++], id), p1++;
    else return Error();

    while(SYM[p1] == 2){
        p1++;
        Graph_init(keywords[2], id);
        if(SYM[p1] == tagtype) Graph_init(ID[p2++], id), p1++;
        else return Error();
    }

    Judge(29);
    Graph_init(keywords[29], id);
    return true;
}

bool WriteStatement(int fa) {
    int id = Graph_init("写语句", fa);
    Judge(27);
    Graph_init(keywords[27], id);
    Judge(28);
    Graph_init(keywords[28], id);
    if(SYM[p1] == tagtype) Graph_init(ID[p2++], id), p1++;
    else return Error();

    while(SYM[p1] == 2){
```

```
        p1++;
        Graph_init(keywords[2], id);
        if(SYM[p1] == tagtype) Graph_init(ID[p2++], id), p1++;
        else return Error();
    }

    Judge(29);
    Graph_init(keywords[29], id);
    return true;
}

bool CompoundStatement(int fa) {
    int id = Graph_init("复合语句", fa);
    Judge(8);
    Graph_init(keywords[8], id);
    if(!Statement(id)) return Error();
    while(SYM[p1] == 3){
        p1++;
        Graph_init(keywords[3], id);
        if(!Statement(id)) return Error();
    }
    Judge(9);
    Graph_init(keywords[9], id);
    return true;
}

/*------------------- Other Functions -------------------*/
bool Item(int fa) {
    int id = Graph_init("项", fa);
    if(!Factor(id)) return Error();
    while(SYM[p1] == 13 || SYM[p1] == 14){
        Graph_init(keywords[SYM[p1++]], id);
        if(!Factor(id)) return Error();
    }
    return true;
}

bool Factor(int fa) {
    int id = Graph_init("因子", fa);
    if(SYM[p1] == tagtype) {
        Graph_init(ID[p2++], id), p1++;
        return true;
    }
    if(SYM[p1] == numtype) {
        Graph_init(ll2string(NUM[p3++]), id), p1++;
        return true;
    }
    if(SYM[p1] == 28) {
```

```
        p1++;
        Graph_init(keywords[28], id);
        if(!Expression(id)) return Error();
        Judge(29);
        Graph_init(keywords[29], id);
        return true;
    }
    return Error();
}

bool Condition(int fa) {
    int id = Graph_init("条件", fa);
    if(SYM[p1] == 10){
        p1++;
        Graph_init(keywords[10], id);
        return Expression(id);
    }
    else{
        if(!Expression(id)) return Error();
        if(SYM[p1] >= 15 && SYM[p1] <= 20){
            Graph_init(keywords[SYM[p1++]], id);
            return Expression(id);
        }
        else return Error();
    }
}

bool Expression(int fa) {
    int id = Graph_init("表达式", fa);
    if(SYM[p1] == 11 || SYM[p1] == 12) Graph_init(keywords[SYM[p1++]], id);
    if(!Item(id)) return Error();
    while(SYM[p1] == 11 || SYM[p1] == 12) {
        Graph_init(keywords[SYM[p1++]], id);
        if(!Item(id)) return Error();
    }
    return true;
}
```

5.2 syntax_tree.py

```
from graphviz import Digraph

def dfs(x, fa, lable, G, graph):
    graph.node(str(x), lable[x])
    if fa != -1:
        graph.edge(str(fa), str(x))
```

```python
    for y in G[x]:
        if y == "":
            break
        dfs(int(y), x, lable, G, graph)


def data_acquire(path):
    # read file
    fo = open(path, "r+")
    data = fo.read().split("\n")

    # acquire data
    n = int(data[0])
    lable = data[1].split(" ")
    G = []
    for row in range(2, len(data)):
        G.append(data[row].split(" "))
    return n, lable, G


if __name__ == "__main__":
    # path list
    path = "PL0_graph.txt"
    out_path1 = "PL0_graph_structure.txt"
    out_path2 = "PL0_graph"

    # init data acquire
    n, lable, G = data_acquire(path)
    # graph = gz.Graph()
    graph = Digraph('G', filename=out_path2)
    graph.attr(rankdir="LR", style="filled", bgcolor="white")
    graph.node_attr.update(style='filled', color='white')
    graph.edge_attr.update(style='filled', color='grey')

    # dfs construct graph
    dfs(int(0), int(-1), lable, G, graph)

    # output graph structure
    fo = open(out_path1, "w")
    fo.write(graph.source)
    graph.render(out_path2, view=True)
```