

## a. 实验目的

本实验旨在设计并实现一个针对自定义小型编程语言的分析器。通过该实验，深入理解程序设计语言的语法结构与词法分析，从而提高对编译原理核心算法的实践能力。

## b. 实验内容

- 设计一个简单的编程语言的语法和词法规则。
- 实现该语言的词法分析器，能够将源代码转换为一系列的记号（tokens）。
- 实现该语言的语法分析器，能够根据词法分析器生成的记号构建抽象语法树（AST）。
- 编写测试用例，验证词法分析器和语法分析器的正确性。

## c. 设计思路

### 语言设计

该实验所虚构的语言具有以下基本特性：

- 支持变量声明和赋值（使用let关键字）。
- 支持 if-else 条件语句。
- 支持 read() 和 write() 输入输出操作。
- 支持基本的算术运算（加、减、乘、除）。
- 支持使用语句块`{}表达式`来表示重复执行的代码段。
- 支持单行注释。

### 词法分析器设计

词法分析器的设计基于正则表达式，用于识别以下记号类型：

- 关键字：let, if, else, read, write
- 标识符：变量名，由字母和数字组成，且以字母开头
- 数字：整数或浮点数
- 运算符：+, -, \*, /, =, ==, !=, <, >, <=, >=, ^^
- 分隔符：;, ,, (), {}

- 注释：以 // 开头的单行注释

## 语法分析器设计

语法分析器采用LL(1)分析方法，主要语法规则详见 e 部分。语法分析器将根据词法分析器生成的记号流，构建抽象语法树（AST）。

### d. 假设条件

- 类型检查：假设  $\wedge$  右侧表达式的结果必须能求值为整数，代表循环次数。
- 声明顺序：假设 let 语句必须出现在变量被赋值或引用之前的物理位置。
- 关键字限制：假设 read 和 print 作为保留字，不能被用作用户定义的变量名。

### e. 相关有限自动机描述

#### 正规表达式

- 标识符  
ID = [A-Za-z\_][A-Za-z0-9\_]\*
- 数字  
NUM = [0-9]+(\.[0-9]+)?
- 注释  
COMMENT = "//" [^\n]\*
- 双字符运算符  
POW2 = "^" EQ = "==" NEQ = "!=" GE = ">=" LE = "<="
- 单字符运算符  
POW1 = "^"  
ASSIGN = "="  
PLUS = "+"  
MINUS = "-"  
MUL = "\*"  
DIV = "/"  
GT = ">"  
LT = "<"
- 单字符分隔符  
SEP = [(){};,]
- 空格  
WS = [\t\n\r]+

## ■ 关键字

KEY = (let|print|read|if|else)

# NFA

---

```
1 ID_NFA:
2   states: {s0, s1}
3   start: s0
4   accept: {s1}
5   transitions:
6     s0 --[A-Zabdcdfghjkmnoqrstuvwxyz_]--> s1
7     s1 --[A-Za-z0-9_]--> s1
8
9 NUM_NFA:
10  states: {s0, s1, s2, s3}
11  start: s0
12  accept: {s1, s3}
13  transitions:
14    s0 --[0-9]--> s1
15    s1 --[0-9]--> s1
16    s1 --[.]--> s2
17    s2 --[0-9]--> s3
18    s3 --[0-9]--> s3
19
20 COMMENT_NFA:
21  states: {s0, s1, s2}
22  start: s0
23  accept: {s2}
24  transitions:
25    s0 --[/]--> s1
26    s1 --[/]--> s2
27    s2 --[^\n]--> s2
28
29 POW2_NFA:
30  states: {s0, s1, s2}
31  start: s0
32  accept: {s2}
33  transitions:
34    s0 --[\^]--> s1
35    s1 --[\^]--> s2
36
37 POW1_NFA:
38  states: {s0, s1}
39  start: s0
40  accept: {s1}
41  transitions:
42    s0 --[\^]--> s1
43
44 EQ_NFA:
45  states: {s0, s1, s2}
46  start: s0
47  accept: {s2}
48  transitions:
49    s0 --[=]--> s1
50    s1 --[=]--> s2
```

```
51
52 NEQ_NFA:
53   states: {s0, s1, s2}
54   start: s0
55   accept: {s2}
56   transitions:
57     s0 --[!]> s1
58     s1 --[=]> s2
59
60 GE_NFA:
61   states: {s0, s1, s2}
62   start: s0
63   accept: {s2}
64   transitions:
65     s0 --[>]> s1
66     s1 --[=]> s2
67
68 LE_NFA:
69   states: {s0, s1, s2}
70   start: s0
71   accept: {s2}
72   transitions:
73     s0 --[<]> s1
74     s1 --[=]> s2
75
76 ASSIGN_NFA:
77   states: {s0, s1}
78   start: s0
79   accept: {s1}
80   transitions:
81     s0 --[=]> s1
82
83 PLUS_NFA:
84   states: {s0, s1}
85   start: s0
86   accept: {s1}
87   transitions:
88     s0 --[+]> s1
89
90 MINUS_NFA:
91   states: {s0, s1}
92   start: s0
93   accept: {s1}
94   transitions:
95     s0 --[-]> s1
96
97 MUL_NFA:
98   states: {s0, s1}
99   start: s0
100  accept: {s1}
101  transitions:
102    s0 --[*]> s1
103
104 DIV_NFA:
105   states: {s0, s1}
106   start: s0
107   accept: {s1}
108   transitions:
```

```

109         s0 --[/]--> s1
110
111 GT_NFA:
112     states: {s0, s1}
113     start: s0
114     accept: {s1}
115     transitions:
116         s0 --[>]--> s1
117
118 LT_NFA:
119     states: {s0, s1}
120     start: s0
121     accept: {s1}
122     transitions:
123         s0 --[<]--> s1
124
125 SEP_NFA:
126     states: {s0, s1}
127     start: s0
128     accept: {s1}
129     transitions:
130         s0 --[(]--> s1
131         s0 --[)]--> s1
132         s0 --[{]--> s1
133         s0 --[}]--> s1
134         s0 --[;]--> s1
135         s0 --[,]--> s1
136
137 WS_NFA:
138     states: {s0, s1}
139     start: s0
140     accept: {s1}
141     transitions:
142         s0 --[ \t\n\r]--> s1
143         s1 --[ \t\n\r]--> s1
144
145 KEY_NFA:
146     states: {s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14}
147     start: s0
148     accept: {s3, ID}
149     transitions:
150
151         s3 --[A-Za-z0-9_]--> ID
152         ID --[A-Za-z0-9_]--> ID
153         # "let"
154         s0 --[l]--> s1
155         s1 --[A-Za-z0-9_]--> ID
156         s1 --[e]--> s2
157         s2 --[A-Za-z0-9_]--> ID
158         s2 --[t]--> s3
159
160         # "print"
161         s0 --[p]--> s4
162         s4 --[A-Za-z0-9_]--> ID
163         s4 --[r]--> s5
164         s5 --[A-Za-z0-9_]--> ID
165         s5 --[i]--> s6
166         s6 --[A-Za-z0-9_]--> ID

```

```

167     s6 --[n]--> s7
168     s7 --[A-Za-su-z0-9_]--> ID
169     s7 --[t]--> s3
170
171     # "read"
172     s0 --[r]--> s9
173     s9 --[A-Za-qst-z0-9_]--> ID
174     s9 --[e]--> s10
175     s10 --[A-Zb-z0-9_]--> ID
176     s10 --[a]--> s11
177     s10 --[A-Za-ce-z0-9_]--> ID
178     s11 --[d]--> s3
179
180     # "if"
181     s0 --[i]--> s13
182     s13 --[A-Za-eg-z0-9_]--> ID
183     s13 --[f]--> s3
184
185     # "else"
186     s0 --[e]--> s15
187     s15 --[A-Za-km-z0-9_]--> ID
188     s15 --[l]--> s16
189     s16 --[A-Za-rt-z0-9_]--> ID
190     s16 --[s]--> s17
191     s17 --[A-Za-df-z0-9_]--> ID
192     s17 --[e]--> s3

```

## 合并

将上述各个 NFA 合并为一个总的 NFA，通过添加一个新的起始状态，并从该状态出发通过  $\varepsilon$  转移到各个子 NFA 的起始状态。最终接受状态为各子 NFA 的接受状态集合。

```

1  LEXER_NFA:
2  states:
3      - S0
4      - all states from each sub-NFA (renamed to avoid collision)
5  start: S0
6  accept:
7      - all accept states from each sub-NFA
8  transitions:
9      # epsilon transitions from global start
10     S0 --[ε]--> ID_s0
11     S0 --[ε]--> NUM_s0
12     S0 --[ε]--> COMMENT_s0
13     S0 --[ε]--> POW2_s0
14     S0 --[ε]--> POW1_s0
15     S0 --[ε]--> EQ_s0
16     S0 --[ε]--> NEQ_s0
17     S0 --[ε]--> GE_s0
18     S0 --[ε]--> LE_s0
19     S0 --[ε]--> ASSIGN_s0
20     S0 --[ε]--> PLUS_s0
21     S0 --[ε]--> MINUS_s0
22     S0 --[ε]--> MUL_s0
23     S0 --[ε]--> DIV_s0

```

```

24 S0 --[ε]--> GT_s0
25 S0 --[ε]--> LT_s0
26 S0 --[ε]--> SEP_s0
27 S0 --[ε]--> WS_s0
28 S0 --[ε]--> KEY_s0
29
30 ID_s0 --[A-Zabdcdfghjkmnoqstuvwxyz_]--> ID_s1
31 ID_s1 --[A-ZZa-z0-9_]--> ID_s1
32
33 NUM_s0 --[0-9]--> NUM_s1
34 NUM_s1 --[0-9]--> NUM_s1
35 NUM_s1 --[.]--> NUM_s2
36 NUM_s2 --[0-9]--> NUM_s3
37 NUM_s3 --[0-9]--> NUM_s3
38
39 COMMENT_s0 --[/]--> COMMENT_s1
40 COMMENT_s1 --[/]--> COMMENT_s2
41 COMMENT_s2 --[^\n]--> COMMENT_s2
42
43 POW2_s0 --[\^]--> POW2_s1
44 POW2_s1 --[\^]--> POW2_s2
45
46 POW1_s0 --[\^]--> POW1_s1
47
48 EQ_s0 --[=]--> EQ_s1
49 EQ_s1 --[=]--> EQ_s2
50
51 NEQ_s0 --[!]--> NEQ_s1
52 NEQ_s1 --[=]--> NEQ_s2
53
54 GE_s0 --[>]--> GE_s1
55 GE_s1 --[=]--> GE_s2
56
57 LE_s0 --[<]--> LE_s1
58 LE_s1 --[=]--> LE_s2
59
60 ASSIGN_s0 --[=]--> ASSIGN_s1
61
62 PLUS_s0 --[+]--> PLUS_s1
63 MINUS_s0 --[-]--> MINUS_s1
64 MUL_s0 --[*]--> MUL_s1
65 DIV_s0 --[/]--> DIV_s1
66 GT_s0 --[>]--> GT_s1
67 LT_s0 --[<]--> LT_s1
68
69 SEP_s0 --[(]--> SEP_s1
70 SEP_s0 --[)]--> SEP_s1
71 SEP_s0 --[{]--> SEP_s1
72 SEP_s0 --[}]--> SEP_s1
73 SEP_s0 --[;]--> SEP_s1
74 SEP_s0 --[, ]--> SEP_s1
75
76 WS_s0 --[ \t\n\r]--> WS_s1
77 WS_s1 --[ \t\n\r]--> WS_s1
78
79 KEY_s3 --[A-Za-z0-9_]--> KEY_ID
80 KEY_ID --[A-Za-z0-9_]--> KEY_ID
81 # "let"

```

```

82     KEY_s0 --[1]--> KEY_s1
83     KEY_s1 --[A-Za-df-z0-9_]--> KEY_ID
84     KEY_s1 --[e]--> KEY_s2
85     KEY_s2 --[A-Za-su-z0-9_]--> KEY_ID
86     KEY_s2 --[t]--> KEY_s3
87
88     # "print"
89     KEY_s0 --[p]--> KEY_s4
90     KEY_s4 --[A-Za-qs-z0-9_]--> KEY_ID
91     KEY_s4 --[r]--> KEY_s5
92     KEY_s5 --[A-Za-hj-z0-9_]--> KEY_ID
93     KEY_s5 --[i]--> KEY_s6
94     KEY_s6 --[A-Za-mo-z0-9_]--> KEY_ID
95     KEY_s6 --[n]--> KEY_s7
96     KEY_s7 --[A-Za-su-z0-9_]--> KEY_ID
97     KEY_s7 --[t]--> KEY_s3
98
99     # "read"
100    KEY_s0 --[r]--> KEY_s9
101    KEY_s9 --[A-Za-qst-z0-9_]--> KEY_ID
102    KEY_s9 --[e]--> KEY_s10
103    KEY_s10 --[A-Zb-z0-9_]--> KEY_ID
104    KEY_s10 --[a]--> KEY_s11
105    KEY_s10 --[A-Za-ce-z0-9_]--> KEY_ID
106    KEY_s11 --[d]--> KEY_s3
107
108    # "if"
109    KEY_s0 --[i]--> KEY_s13
110    KEY_s13 --[A-Za-eg-z0-9_]--> KEY_ID
111    KEY_s13 --[f]--> KEY_s3
112
113    # "else"
114    KEY_s0 --[e]--> KEY_s15
115    KEY_s15 --[A-Za-km-z0-9_]--> KEY_ID
116    KEY_s15 --[l]--> KEY_s16
117    KEY_s16 --[A-Za-rt-z0-9_]--> KEY_ID
118    KEY_s16 --[s]--> KEY_s17
119    KEY_s17 --[A-Za-df-z0-9_]--> KEY_ID
120    KEY_s17 --[e]--> KEY_s3

```

## 最小化

通过状态合并和消除不可达状态，对上述 NFA 进行最小化处理，得到一个等价的最小化 DFA，用于实际的词法分析过程。表示为json格式以便于程序读取和处理。

```

1  {
2      "states": [
3          "START",
4          "ID",
5          "NUM_INT",
6          "NUM_MIDDLE",
7          "NUM_FLOAT",
8          "OP1",
9          "OP2",
10         "REPEAT",

```



```
11     "ASSIGN",
12     "WS",
13     "SEP",
14     "COMMENT",
15     "KEY"
16 ],
17 "start": "START",
18 "accept": [
19     "ID",
20     "NUM_INT",
21     "NUM_FLOAT",
22     "OP1",
23     "OP2",
24     "REPEAT",
25     "ASSIGN",
26     "WS",
27     "SEP",
28     "COMMENT",
29     "KEY"
30 ],
31 "trans": [
32     {
33         "from": "START",
34         "to": "ID",
35         "pattern": "[A-Zabcdefghijklmnopqrstuvwxyz_]"
36     },
37     {
38         "from": "ID",
39         "to": "ID",
40         "pattern": "[A-Za-z0-9_]"
41     },
42     {
43         "from": "START",
44         "to": "NUM_INT",
45         "pattern": "[0-9]"
46     },
47     {
48         "from": "NUM_INT",
49         "to": "NUM_INT",
50         "pattern": "[0-9]"
51     },
52     {
53         "from": "NUM_INT",
54         "to": "NUM_MIDDLE",
55         "pattern": "[.]"
56     },
57     {
58         "from": "NUM_MIDDLE",
59         "to": "NUM_FLOAT",
60         "pattern": "[0-9]"
61     },
62     {
63         "from": "NUM_FLOAT",
64         "to": "NUM_FLOAT",
65         "pattern": "[0-9]"
66     },
67     {
68         "from": "START",
```

```

69         "to": "OP1",
70         "pattern": "[\\^]"
71     },
72     {
73         "from": "OP1",
74         "to": "REPEAT",
75         "pattern": "[\\^]"
76     },
77     {
78         "from": "START",
79         "to": "ASSIGN",
80         "pattern": "[=]"
81     },
82     {
83         "from": "ASSIGN",
84         "to": "OP2",
85         "pattern": "[=]"
86     },
87     {
88         "from": "START",
89         "to": "OP1",
90         "pattern": "[<]"
91     },
92     {
93         "from": "OP1",
94         "to": "OP2",
95         "pattern": "[=]"
96     },
97     {
98         "from": "START",
99         "to": "OP1",
100        "pattern": "[/]"
101    },
102    {
103        "from": "START",
104        "to": "OP1",
105        "pattern": "[>]"
106    },
107    {
108        "from": "OP1",
109        "to": "OP2",
110        "pattern": "[=]"
111    },
112    {
113        "from": "START",
114        "to": "OP1",
115        "pattern": "[-]"
116    },
117    {
118        "from": "START",
119        "to": "OP1",
120        "pattern": "[+]"
121    },
122    {
123        "from": "START",
124        "to": "OP1",
125        "pattern": "[*]"
126    },

```

```
127     {
128         "from": "START",
129         "to": "WS",
130         "pattern": "[ \\t\\n\\r]"
131     },
132     {
133         "from": "WS",
134         "to": "WS",
135         "pattern": "[ \\t\\n\\r]"
136     },
137     {
138         "from": "START",
139         "to": "OP1",
140         "pattern": "[!]"
141     },
142     {
143         "from": "OP1",
144         "to": "OP2",
145         "pattern": "[=]"
146     },
147     {
148         "from": "START",
149         "to": "SEP",
150         "pattern": "[(){};,]"
151     },
152     {
153         "from": "OP1",
154         "to": "COMMENT",
155         "pattern": "[/]"
156     },
157     {
158         "from": "COMMENT",
159         "to": "COMMENT",
160         "pattern": "[^\\n]"
161     },
162     {
163         "from": "KEY",
164         "to": "ID",
165         "pattern": "[A-Za-z0-9_]"
166     },
167     {
168         "from": "START",
169         "to": "LET:L",
170         "pattern": "[1]"
171     },
172     {
173         "from": "LET:L",
174         "to": "ID",
175         "pattern": "[A-Za-z0-9_]"
176     },
177     {
178         "from": "LET:L",
179         "to": "LET:E",
180         "pattern": "[e]"
181     },
182     {
183         "from": "LET:E",
184         "to": "ID",
```

```
185         "pattern": "[A-Za-su-z0-9_]"
186     },
187     {
188         "from": "LET:E",
189         "to": "KEY",
190         "pattern": "[t]"
191     },
192     {
193         "from": "START",
194         "to": "PRINT:P",
195         "pattern": "[p]"
196     },
197     {
198         "from": "PRINT:P",
199         "to": "ID",
200         "pattern": "[A-Za-qs-z0-9_]"
201     },
202     {
203         "from": "PRINT:P",
204         "to": "PRINT:R",
205         "pattern": "[r]"
206     },
207     {
208         "from": "PRINT:R",
209         "to": "ID",
210         "pattern": "[A-Za-hj-z0-9_]"
211     },
212     {
213         "from": "PRINT:R",
214         "to": "PRINT:I",
215         "pattern": "[i]"
216     },
217     {
218         "from": "PRINT:I",
219         "to": "ID",
220         "pattern": "[A-Za-mo-z0-9_]"
221     },
222     {
223         "from": "PRINT:I",
224         "to": "PRINT:N",
225         "pattern": "[n]"
226     },
227     {
228         "from": "PRINT:N",
229         "to": "ID",
230         "pattern": "[A-Za-su-z0-9_]"
231     },
232     {
233         "from": "PRINT:N",
234         "to": "KEY",
235         "pattern": "[t]"
236     },
237     {
238         "from": "START",
239         "to": "READ:R",
240         "pattern": "[r]"
241     },
242     {
```

```
243     "from": "READ:R",
244     "to": "ID",
245     "pattern": "[A-Za-df-z0-9_]"
246 },
247 {
248     "from": "READ:R",
249     "to": "READ:E",
250     "pattern": "[e]"
251 },
252 {
253     "from": "READ:E",
254     "to": "ID",
255     "pattern": "[A-Zb-z0-9_]"
256 },
257 {
258     "from": "READ:E",
259     "to": "READ:A",
260     "pattern": "[a]"
261 },
262 {
263     "from": "READ:A",
264     "to": "ID",
265     "pattern": "[A-Za-ce-z0-9_]"
266 },
267 {
268     "from": "READ:A",
269     "to": "KEY",
270     "pattern": "[d]"
271 },
272 {
273     "from": "START",
274     "to": "IF:I",
275     "pattern": "[i]"
276 },
277 {
278     "from": "IF:I",
279     "to": "ID",
280     "pattern": "[A-Za-eg-z0-9_]"
281 },
282 {
283     "from": "IF:I",
284     "to": "KEY",
285     "pattern": "[f]"
286 },
287 {
288     "from": "START",
289     "to": "ELSE:E",
290     "pattern": "[e]"
291 },
292 {
293     "from": "ELSE:E",
294     "to": "ID",
295     "pattern": "[A-Za-km-z0-9_]"
296 },
297 {
298     "from": "ELSE:E",
299     "to": "ELSE:L",
300     "pattern": "[l]"
```

```

301     },
302     {
303         "from": "ELSE:L",
304         "to": "ID",
305         "pattern": "[A-Za-rt-z0-9_]"
306     },
307     {
308         "from": "ELSE:L",
309         "to": "ELSE:S",
310         "pattern": "[s]"
311     },
312     {
313         "from": "ELSE:S",
314         "to": "ID",
315         "pattern": "[A-Za-df-z0-9_]"
316     },
317     {
318         "from": "ELSE:S",
319         "to": "KEY",
320         "pattern": "[e]"
321     }
322 ]
323 }

```

## CFG

---

- Program  $\rightarrow$  StmtList
- StmtList  $\rightarrow$  Stmt StmtList  $\mid \varepsilon$
- Stmt  $\rightarrow$  DeclStmt  $\mid$  AssignStmt  $\mid$  BlockMaybeRepeat  $\mid$  FuncCallStmt  $\mid$  IfStmt
- DeclStmt  $\rightarrow$  KEY(let) ID SEP(;
- AssignStmt  $\rightarrow$  ID ASSIGN Expr SEP(;
- BlockMaybeRepeat  $\rightarrow$  Block  $\mid$  RepeatTail
- Block  $\rightarrow$  SEP({) StmtList SEP(})
- FuncCallStmt  $\rightarrow$  FuncCall SEP(;
- FuncCall  $\rightarrow$  KEY(print) SEP() Expr SEP())  $\mid$  KEY(read) SEP() ID SEP())
- IfStmt  $\rightarrow$  KEY(if) SEP() Expr SEP()) Block ElsePart
- ElsePart  $\rightarrow$  KEY(else) Block  $\mid \varepsilon$
- RepeatTail  $\rightarrow$  REPEAT Expr SEP(;)  $\mid \varepsilon$
- Expr  $\rightarrow$  EqualityExpr
- EqualityExpr  $\rightarrow$  RelationExpr EqualityExpr'
- EqualityExpr'  $\rightarrow$  OP2(==) RelationExpr EqualityExpr'  $\mid$  OP2(!=) RelationExpr EqualityExpr'  $\mid \varepsilon$
- RelationExpr  $\rightarrow$  AddExpr RelationExpr'
- RelationExpr'  $\rightarrow$  OP2(>=) AddExpr RelationExpr'  $\mid$  OP2(<=) AddExpr RelationExpr'  $\mid$  OP1(>) AddExpr RelationExpr'  $\mid$  OP1(<) AddExpr RelationExpr'  $\mid \varepsilon$
- AddExpr  $\rightarrow$  MulExpr AddExpr'
- AddExpr'  $\rightarrow$  OP1(+) MulExpr AddExpr'  $\mid$  OP1(-) MulExpr AddExpr'  $\mid \varepsilon$

- $\text{MulExpr} \rightarrow \text{UnaryExpr MulExpr}'$
- $\text{MulExpr}' \rightarrow \text{OP1}(\ast) \text{UnaryExpr MulExpr}' \mid \text{OP1}(/) \text{UnaryExpr MulExpr}' \mid \epsilon$
- $\text{UnaryExpr} \rightarrow \text{OP1}(+) \text{UnaryExpr} \mid \text{OP1}(-) \text{UnaryExpr} \mid \text{OP1}(!) \text{UnaryExpr} \mid \text{PowerExpr}$
- $\text{PowerExpr} \rightarrow \text{PrimaryExpr PowerExpr}'$
- $\text{PowerExpr}' \rightarrow \text{OP1}(\wedge) \text{PowerExpr}' \mid \epsilon$
- $\text{PrimaryExpr} \rightarrow \text{SEP}() \text{Expr SEP}() \mid \text{NUM\_INT} \mid \text{NUM\_FLOAT} \mid \text{ID}$

## First and Follow Sets

- First Sets
  - $\text{First}(\text{Program}) = \text{First}(\text{StmtList}) = \{\text{KEY}(\text{let}), \text{ID}, \text{SEP}(\{\}), \text{KEY}(\text{print}), \text{KEY}(\text{read}), \text{KEY}(\text{if}), \epsilon\}$
  - $\text{First}(\text{Stmt}) = \{\text{KEY}(\text{let}), \text{ID}, \text{SEP}(\{\}), \text{KEY}(\text{print}), \text{KEY}(\text{read}), \text{KEY}(\text{if})\}$
  - $\text{First}(\text{DeclStmt}) = \{\text{KEY}(\text{let})\}$
  - $\text{First}(\text{AssignStmt}) = \{\text{ID}\}$
  - $\text{First}(\text{BlockMaybeRepeat}) = \text{First}(\text{Block}) = \{\text{SEP}(\{\})\}$
  - $\text{First}(\text{FuncCallStmt}) = \text{First}(\text{FuncCall}) = \{\text{KEY}(\text{print}), \text{KEY}(\text{read})\}$
  - $\text{First}(\text{IfStmt}) = \{\text{KEY}(\text{if})\}$
  - $\text{First}(\text{ElsePart}) = \{\text{KEY}(\text{else}), \epsilon\}$
  - $\text{First}(\text{RepeatTail}) = \{\text{REPEAT}, \epsilon\}$
  - $\text{First}(\text{Expr}) = \text{First}(\text{EqualityExpr}) = \text{First}(\text{RelationExpr}) = \text{First}(\text{AddExpr}) = \text{First}(\text{MulExpr}) = \{\text{SEP}() , \text{NUM\_INT}, \text{NUM\_FLOAT}, \text{ID}, \text{OP1}(+), \text{OP1}(-), \text{OP1}(!)\}$
  - $\text{First}(\text{UnaryExpr}) = \{\text{OP1}(+), \text{OP1}(-), \text{OP1}(!), \text{SEP}() , \text{NUM\_INT}, \text{NUM\_FLOAT}, \text{ID}\}$
  - $\text{First}(\text{EqualityExpr}') = \{\text{OP2}(==), \text{OP2}(!=), \epsilon\}$
  - $\text{First}(\text{RelationExpr}') = \{\text{OP2}(>=), \text{OP2}(<=), \text{OP1}(>), \text{OP1}(<), \epsilon\}$
  - $\text{First}(\text{AddExpr}') = \{\text{OP1}(+), \text{OP1}(-), \epsilon\}$
  - $\text{First}(\text{MulExpr}') = \{\text{OP1}(\ast), \text{OP1}(/), \epsilon\}$
  - $\text{First}(\text{PowerExpr}) = \{\text{SEP}() , \text{NUM\_INT}, \text{NUM\_FLOAT}, \text{ID}\}$
  - $\text{First}(\text{PowerExpr}') = \{\text{OP1}(\wedge), \epsilon\}$
  - $\text{First}(\text{PrimaryExpr}) = \{\text{SEP}() , \text{NUM\_INT}, \text{NUM\_FLOAT}, \text{ID}\}$
- Follow Sets
  - $\text{Follow}(\text{Program}) = \{\#\}$
  - $\text{Follow}(\text{StmtList}) = \{\#, \text{SEP}(\{\})\}$
  - $\text{Follow}(\text{Stmt}) = \text{Follow}(\text{DeclStmt}) = \text{Follow}(\text{AssignStmt}) = \text{Follow}(\text{IfStmt}) = \text{Follow}(\text{ElsePart}) = \text{Follow}(\text{Block}) = \{\text{KEY}(\text{let}), \text{ID}, \text{SEP}(\{\}), \text{KEY}(\text{print}), \text{KEY}(\text{read}), \text{KEY}(\text{if}), \text{SEP}(\{\}), \#\}$
  - $\text{Follow}(\text{BlockMaybeRepeat}) = \{\text{KEY}(\text{let}), \text{ID}, \text{SEP}(\{\}), \text{KEY}(\text{print}), \text{KEY}(\text{read}), \text{KEY}(\text{if}), \text{SEP}(\{\}), \#, \text{KEY}(\text{else}), \text{REPEAT}\}$
  - $\text{Follow}(\text{RepeatTail}) = \{\text{KEY}(\text{let}), \text{ID}, \text{SEP}(\{\}), \text{KEY}(\text{print}), \text{KEY}(\text{read}), \text{KEY}(\text{if}), \text{SEP}(\{\}), \#\}$
  - $\text{Follow}(\text{FuncCallStmt}) = \text{Follow}(\text{FuncCall}) = \{\text{SEP}(\{;})\}$
  - $\text{Follow}(\text{Expr}) = \text{Follow}(\text{EqualityExpr}) = \text{Follow}(\text{EqualityExpr}') = \{\text{SEP}(), \text{SEP}(\{;})\}$
  - $\text{Follow}(\text{RelationExpr}) = \text{Follow}(\text{RelationExpr}') = \{\text{OP2}(==), \text{OP2}(!=), \text{SEP}(), \text{SEP}(\{;})\}$

- Follow(AddExpr) = Follow(AddExpr') = {OP2(>=), OP2(<=), OP1(>), OP1(<), OP2(==), OP2(!=), SEP()), SEP(;)}
  - Follow(MulExpr) = Follow(MulExpr') = {OP1(+), OP1(-), OP2(>=), OP2(<=), OP1(>), OP1(<), OP2(==), OP2(!=), SEP()), SEP(;)}
  - Follow(UnaryExpr) = {OP1(\*), OP1(/), OP1(+), OP1(-), OP2(>=), OP2(<=), OP1(>), OP1(<), OP2(==), OP2(!=), SEP()), SEP(;)}
  - Follow(PowerExpr) = Follow(PowerExpr') = {OP1(^), OP1(\*), OP1(/), OP1(+), OP1(-), OP2(>=), OP2(<=), OP1(>), OP1(<), OP2(==), OP2(!=), SEP()), SEP(;)}
  - Follow(PrimaryExpr) = {OP1(^), OP1(\*), OP1(/), OP1(+), OP1(-), OP2(>=), OP2(<=), OP1(>), OP1(<), OP2(==), OP2(!=), SEP()), SEP(;)}

## LL(1) Parsing Table

注：此文档由markdown编写，转化为pdf后此表可能展示不全，如需所有详细信息，请阅读report.md文件源码

Non-Terminal	KEY(let)	ID	SEP()	KEY(print)	KEY(read)	KEY(if)	SEP()	NUM_INT	NUM_FLOAT	OP1(+)	OP1(-)	OP1(!)	SEP()	KEY(else)	REPEAT	SEP()	#
Program	StmtList	StmtList	StmtList	StmtList	StmtList	StmtList											StmtList
StmtList	Stmt	Stmt	Stmt	Stmt	Stmt	Stmt							ε			ε	ε
Stmt	DeclStmt	AssignStmt	BlockMaybeRepeat	FuncCallStmt	FuncCallStmt	IfStmt											
DeclStmt	KEY(let) ID	SEP()															
AssignStmt		ID ASSIGN	Expr SEP()														
BlockMaybeRepeat			Block RepeatTail														
RepeatTail															REPEAT	Expr SEP()	ε
Block			SEP() StmtList	SEP()													
FuncCallStmt				FuncCall SEP()	FuncCall SEP()												
FuncCall				KEY(print) SEP()	KEY(read) SEP()	ID SEP()											
IfStmt						KEY(if) SEP()	Expr SEP()	Block	ElsePart								
ElsePart													ε	KEY(else)	Block		
Expr				SEP()	Expr	NUM_INT	Expr	NUM_FLOAT	Expr	OP1(+)	Expr	OP1(-)	Expr	OP1(!)	Expr		
EqualityExpr				SEP()	RelationExpr	EqualityExpr'		NUM_INT	RelationExpr	EqualityExpr'		NUM_FLOAT	RelationExpr	EqualityExpr'			
EqualityExpr'															ε		ε
RelationExpr				SEP()	AddExpr	RelationExpr'		NUM_INT	AddExpr	RelationExpr'		NUM_FLOAT	AddExpr	RelationExpr'			
RelationExpr'															ε		ε
AddExpr				SEP()	MulExpr	AddExpr'		NUM_INT	MulExpr	AddExpr'		NUM_FLOAT	MulExpr	AddExpr'			
AddExpr'															ε		ε
MulExpr				SEP()	UnaryExpr	MulExpr'		NUM_INT	UnaryExpr	MulExpr'		NUM_FLOAT	UnaryExpr	MulExpr'			
MulExpr'															ε		ε
UnaryExpr				SEP()	PowerExpr			NUM_INT	PowerExpr			NUM_FLOAT	PowerExpr				
PowerExpr				SEP()	PrimaryExpr	PowerExpr'		NUM_INT	PrimaryExpr	PowerExpr'		NUM_FLOAT	PrimaryExpr	PowerExpr'			
PowerExpr'															ε		ε
PrimaryExpr				SEP()	Expr SEP()			NUM_INT				NUM_FLOAT					

## f. 重要数据结构描述

### Token

```
1 #[derive(Debug, Clone, PartialEq, Eq, Hash, Serialize)]
2 pub struct Token {
3     pub kind: String,
4     pub lexeme: String,
```



Token 是词法分析阶段的产物，也是语法分析阶段的最小输入单位。

成员：

kind: 类别标识（如 "ID", "NUM", "IF"），在语法分析中作为终结符匹配。

lexeme: 源代码中的原始文本内容（如变量名 `x` 或数字 `3.14`）。

关键逻辑：

to\_key() 方法：这是一个映射逻辑，用于在 LL(1) 查表时将具体的 lexeme 归类。例如，所有的标识符在查表时都统一视为 "ID"。

## DFA

```
1  #[derive(Debug, Deserialize)]
2  struct DFA {
3      states: Vec<String>,
4      start: String,
5      accept: Vec<String>,
6      trans: Vec<Transition>,
7  }
```

该结构用于描述如何从字符串流中切割出 Token。

成员：

states: 状态集合。

start: 起始状态。

accept: 接受状态列表。这里的关键设计是：lex\_code 函数将到达的接受状态名称直接作为生成的 Token 的 kind。

trans: 转移函数集合，定义了在当前状态下遇到特定字符（pattern）时应跳转到哪个状态。

## L1Table

```
1  #[derive(Debug)]
2  struct L1Table {
3      terminals: Vec<Token>,
4      non_terminals: Vec<Token>,
5      table: HashMap<Token, HashMap<Token, Vec<Token>>>,
6  }
```

这是语法分析器的“大脑”，决定了给定当前的非终结符和输入符号，应该推导出哪条产生式。

结构：HashMap<Token, HashMap<Token, Vec>>

外层 Key: 当前栈顶的非终结符（Non-terminal）。

内层 Key: 当前输入的终结符（Terminal/Lookahead）。

Value: 对应的产生式右部（由 Token 组成的向量）。

特殊符号：支持  $\epsilon$ （空产生式）和 #（输入结束标记）。

## ASTNode (抽象语法树节点)

```
1  #[derive(Debug, Clone)]
2  pub enum ASTNode {
3      Terminal {
4          token: Token,
5      },
6      NonTerminal {
7          kind: String,
8          children: Vec<Rc<RefCell<ASTNode>>>,
9      },
10 }
```

这是一个递归的枚举类型，代表了源代码的树形结构化表示。

变体：

Terminal: 叶子节点，存储具体的 Token。

NonTerminal: 中间节点，包含一个类别名（kind）和一组子节点（children）。

设计细节：

使用了 Rc<RefCell>。这是为了在 LL(1) 解析这种自顶向下的过程中，能够方便地在父节点创建后动态地追加子节点，解决 Rust 的所有权和可变性限制。

# g. 核心算法描述

## 词法分析：基于 DFA 的最大匹配逻辑

lex\_code 函数是程序的第一道工序，它负责将源代码切割成一个个 Token。

状态转移与贪婪匹配：程序从 pos 位置开始，模拟 DFA 的状态跳转。它不仅记录当前是否到达了 accept（接受）状态，还会尽可能向后多读字符（last\_accept\_pos）。

回溯机制：如果读到某个字符发现无法继续跳转，它会“回退”到最后一个记录的接受状态。

Token 产出：一旦确定了最长匹配，就将该段文本截取，并以 DFA 状态名作为 Token 的 kind。

## 语法分析：基于栈的 LL(1) 驱动算法

parse\_ll1 函数是程序最复杂的逻辑所在。它实现了一个非递归的预测分析器，核心逻辑是一个 while 循环：

符号匹配（Match）：

如果栈顶是终结符（Terminal），且与当前输入的 Token 种类一致，则匹配成功，同步弹出栈顶和输入流头部的符号。

展开产生式（Expand）：

如果栈顶是非终结符（Non-Terminal），程序会去 L11Table 中查表：table[栈顶符号][当前输入符号]。

查到的结果是一条产生式（如  $A \rightarrow BC$ ）。程序会将栈顶的 A 弹出，然后将右部的符号 C 和 B 逆序压入栈中（这样 B 就在栈顶，能够先被处理）。

空产生式处理：如果查到的是  $\epsilon$ （epsilon），表示该非终结符可以推导为空，此时只需弹出栈顶，不压入新符号。

# AST 的动态构建逻辑

由于使用的是非递归分析，如何在“展开”产生式的同时构建树状结构是一个难点。代码通过以下方式解决：

同步对象：在将符号压入栈时，不仅压入了符号名，还压入了一个 Rc<RefCell>。

父子绑定：

当一个非终结符被展开时，它会先创建好所有子节点的“空壳”。

这些子节点被存入父节点的 children 向量中。

同时，这些子节点被压入栈。当后续逻辑处理到这些栈元素时，会直接填充这些“空壳”的内容（如填充 Terminal 的 lexeme）。

引用计数管理：通过 Rc<RefCell<...>>，父节点和栈同时拥有指向子节点的指针，确保了在复杂的压栈/弹栈过程中，树形结构能被正确连接。

# h. 运行用例

测试源代码(./test\_code.txt)为

```
1 let x;
2 x = 42;
3 let y;
4 read(y);
5 let abc;
6 abc = x + y + 23.3;
7 {
8     abc = abc * 23.3;
9 } ^^ 66;
10 if (x >= y + x == y + x != y) {print(x);} else {print(y);}
```

运行程序后，输出的 Token 序列(./dfa\_lexer/token.txt)为：

1	AcceptState: KEY	Lexeme: 'let'
2	AcceptState: WS	Lexeme: ' '
3	AcceptState: ID	Lexeme: 'x'
4	AcceptState: SEP	Lexeme: ';'

5	AcceptState: WS	Lexeme: '\n'
6	AcceptState: ID	Lexeme: 'x'
7	AcceptState: WS	Lexeme: ' '
8	AcceptState: ASSIGN	Lexeme: '='
9	AcceptState: WS	Lexeme: ' '
10	AcceptState: NUM_INT	Lexeme: '42'
11	AcceptState: SEP	Lexeme: ';'
12	AcceptState: WS	Lexeme: '\n'
13	AcceptState: KEY	Lexeme: 'let'
14	AcceptState: WS	Lexeme: ' '
15	AcceptState: ID	Lexeme: 'y'
16	AcceptState: SEP	Lexeme: ';'
17	AcceptState: WS	Lexeme: '\n'
18	AcceptState: KEY	Lexeme: 'read'
19	AcceptState: SEP	Lexeme: '('
20	AcceptState: ID	Lexeme: 'y'
21	AcceptState: SEP	Lexeme: ')'
22	AcceptState: SEP	Lexeme: ';'
23	AcceptState: WS	Lexeme: '\n'
24	AcceptState: KEY	Lexeme: 'let'
25	AcceptState: WS	Lexeme: ' '
26	AcceptState: ID	Lexeme: 'abc'
27	AcceptState: SEP	Lexeme: ';'
28	AcceptState: WS	Lexeme: '\n'
29	AcceptState: ID	Lexeme: 'abc'
30	AcceptState: WS	Lexeme: ' '
31	AcceptState: ASSIGN	Lexeme: '='
32	AcceptState: WS	Lexeme: ' '
33	AcceptState: ID	Lexeme: 'x'
34	AcceptState: WS	Lexeme: ' '
35	AcceptState: OP1	Lexeme: '+'
36	AcceptState: WS	Lexeme: ' '
37	AcceptState: ID	Lexeme: 'y'
38	AcceptState: WS	Lexeme: ' '
39	AcceptState: OP1	Lexeme: '+'
40	AcceptState: WS	Lexeme: ' '
41	AcceptState: NUM_FLOAT	Lexeme: '23.3'
42	AcceptState: SEP	Lexeme: ';'
43	AcceptState: WS	Lexeme: '\n'
44	AcceptState: SEP	Lexeme: '{'
45	AcceptState: WS	Lexeme: '\n      '
46	AcceptState: ID	Lexeme: 'abc'
47	AcceptState: WS	Lexeme: ' '
48	AcceptState: ASSIGN	Lexeme: '='
49	AcceptState: WS	Lexeme: ' '
50	AcceptState: ID	Lexeme: 'abc'
51	AcceptState: WS	Lexeme: ' '
52	AcceptState: OP1	Lexeme: '*'
53	AcceptState: WS	Lexeme: ' '
54	AcceptState: NUM_FLOAT	Lexeme: '23.3'
55	AcceptState: SEP	Lexeme: ';'
56	AcceptState: WS	Lexeme: '\n'
57	AcceptState: SEP	Lexeme: '}'
58	AcceptState: WS	Lexeme: ' '
59	AcceptState: REPEAT	Lexeme: '^'
60	AcceptState: WS	Lexeme: ' '
61	AcceptState: NUM_INT	Lexeme: '66'
62	AcceptState: SEP	Lexeme: ';'

63	AcceptState: WS	Lexeme: '\n'
64	AcceptState: KEY	Lexeme: 'if'
65	AcceptState: WS	Lexeme: ' '
66	AcceptState: SEP	Lexeme: '('
67	AcceptState: ID	Lexeme: 'x'
68	AcceptState: WS	Lexeme: ' '
69	AcceptState: OP2	Lexeme: '>='
70	AcceptState: WS	Lexeme: ' '
71	AcceptState: ID	Lexeme: 'y'
72	AcceptState: WS	Lexeme: ' '
73	AcceptState: OP1	Lexeme: '+'
74	AcceptState: WS	Lexeme: ' '
75	AcceptState: ID	Lexeme: 'x'
76	AcceptState: WS	Lexeme: ' '
77	AcceptState: OP2	Lexeme: '=='
78	AcceptState: WS	Lexeme: ' '
79	AcceptState: ID	Lexeme: 'y'
80	AcceptState: WS	Lexeme: ' '
81	AcceptState: OP1	Lexeme: '+'
82	AcceptState: WS	Lexeme: ' '
83	AcceptState: ID	Lexeme: 'x'
84	AcceptState: WS	Lexeme: ' '
85	AcceptState: OP2	Lexeme: '!='
86	AcceptState: WS	Lexeme: ' '
87	AcceptState: ID	Lexeme: 'y'
88	AcceptState: SEP	Lexeme: ')'
89	AcceptState: WS	Lexeme: ' '
90	AcceptState: SEP	Lexeme: '{'
91	AcceptState: KEY	Lexeme: 'print'
92	AcceptState: SEP	Lexeme: '('
93	AcceptState: ID	Lexeme: 'x'
94	AcceptState: SEP	Lexeme: ')'
95	AcceptState: SEP	Lexeme: ';'
96	AcceptState: SEP	Lexeme: '}'
97	AcceptState: WS	Lexeme: ' '
98	AcceptState: KEY	Lexeme: 'else'
99	AcceptState: WS	Lexeme: ' '
100	AcceptState: SEP	Lexeme: '{'
101	AcceptState: KEY	Lexeme: 'print'
102	AcceptState: SEP	Lexeme: '('
103	AcceptState: ID	Lexeme: 'y'
104	AcceptState: SEP	Lexeme: ')'
105	AcceptState: SEP	Lexeme: ';'
106	AcceptState: SEP	Lexeme: '}'

最后，生成的抽象语法树（AST）（./dfa\_lexer/ast.json）如下所示：

```

1  {
2    "type": "NonTerminal",
3    "kind": "Program",
4    "children": [
5      {
6        "type": "NonTerminal",
7        "kind": "StmtList",
8        "children": [
9          {
10         "type": "NonTerminal",

```

```

11     "kind": "Stmt",
12     "children": [
13         {
14             "type": "NonTerminal",
15             "kind": "DeclStmt",
16             "children": [
17                 {
18                     "kind": "Terminal",
19                     "token": {
20                         "kind": "KEY",
21                         "lexeme": "let"
22                     }
23                 },
24                 {
25                     "kind": "Terminal",
26                     "token": {
27                         "kind": "ID",
28                         "lexeme": "x"
29                     }
30                 },
31                 {
32                     "kind": "Terminal",
33                     "token": {
34                         "kind": "SEP",
35                         "lexeme": ";"
36                     }
37                 }
38             ]
39         }
40     ]
41 },
42 {
43     "type": "NonTerminal",
44     "kind": "StmtList",
45     "children": [
46         {
47             "type": "NonTerminal",
48             "kind": "Stmt",
49             "children": [
50                 {
51                     "type": "NonTerminal",
52                     "kind": "AssignStmt",
53                     "children": [
54                         {
55                             "kind": "Terminal",
56                             "token": {
57                                 "kind": "ID",
58                                 "lexeme": "x"
59                             }
60                         },
61                         {
62                             "kind": "Terminal",
63                             "token": {
64                                 "kind": "ASSIGN",
65                                 "lexeme": "="
66                             }
67                         },
68                     ]

```

```

69     "type": "NonTerminal",
70     "kind": "Expr",
71     "children": [
72     {
73         "type": "NonTerminal",
74         "kind": "EqualityExpr",
75         "children": [
76         {
77             "type": "NonTerminal",
78             "kind": "RelationExpr",
79             "children": [
80             {
81                 "type": "NonTerminal",
82                 "kind": "AddExpr",
83                 "children": [
84                 {
85                     "type": "NonTerminal",
86                     "kind": "MulExpr",
87                     "children": [
88                     {
89                         "type": "NonTerminal",
90                         "kind": "UnaryExpr",
91                         "children": [
92                         {
93                             "type": "NonTerminal",
94                             "kind": "PowerExpr",
95                             "children": [
96                             {
97                                 "type": "NonTerminal",
98                                 "kind": "PrimaryExpr",
99                                 "children": [
100                                {
101                                    "kind": "Terminal",
102                                    "token": {
103                                        "kind": "NUM_INT",
104                                        "lexeme": "42"
105                                    }
106                                }
107                            ]
108                        },
109                        {
110                            "type": "NonTerminal",
111                            "kind": "PowerExpr'",
112                            "children": []
113                        }
114                    ]
115                }
116            ]
117            },
118            {
119                "type": "NonTerminal",
120                "kind": "MulExpr'",
121                "children": []
122            }
123        ]
124        },
125        {
126            "type": "NonTerminal",

```

[illegible]



```

185         {
186             "kind": "Terminal",
187             "token": {
188                 "kind": "SEP",
189                 "lexeme": ";"
190             }
191         }
192     ]
193 }
194 ]
195 },
196 {
197     "type": "NonTerminal",
198     "kind": "StmtList",
199     "children": [
200     {
201         "type": "NonTerminal",
202         "kind": "Stmt",
203         "children": [
204         {
205             "type": "NonTerminal",
206             "kind": "FuncCallStmt",
207             "children": [
208             {
209                 "type": "NonTerminal",
210                 "kind": "FuncCall",
211                 "children": [
212                 {
213                     "kind": "Terminal",
214                     "token": {
215                         "kind": "KEY",
216                         "lexeme": "read"
217                     }
218                 },
219                 {
220                     "kind": "Terminal",
221                     "token": {
222                         "kind": "SEP",
223                         "lexeme": "("
224                     }
225                 },
226                 {
227                     "kind": "Terminal",
228                     "token": {
229                         "kind": "ID",
230                         "lexeme": "y"
231                     }
232                 },
233                 {
234                     "kind": "Terminal",
235                     "token": {
236                         "kind": "SEP",
237                         "lexeme": ")"
238                     }
239                 }
240             ]
241         },
242         {

```

```

243         "kind": "Terminal",
244         "token": {
245             "kind": "SEP",
246             "lexeme": ";"
247         }
248     }
249 ]
250 }
251 ]
252 },
253 {
254     "type": "NonTerminal",
255     "kind": "StmtList",
256     "children": [
257     {
258         "type": "NonTerminal",
259         "kind": "Stmt",
260         "children": [
261         {
262             "type": "NonTerminal",
263             "kind": "DeclStmt",
264             "children": [
265             {
266                 "kind": "Terminal",
267                 "token": {
268                     "kind": "KEY",
269                     "lexeme": "let"
270                 }
271             },
272             {
273                 "kind": "Terminal",
274                 "token": {
275                     "kind": "ID",
276                     "lexeme": "abc"
277                 }
278             },
279             {
280                 "kind": "Terminal",
281                 "token": {
282                     "kind": "SEP",
283                     "lexeme": ";"
284                 }
285             }
286         ]
287         }
288     ]
289 },
290 {
291     "type": "NonTerminal",
292     "kind": "StmtList",
293     "children": [
294     {
295         "type": "NonTerminal",
296         "kind": "Stmt",
297         "children": [
298         {
299             "type": "NonTerminal",
300             "kind": "AssignStmt",

```

301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358

```

children": [
{
  "kind": "Terminal",
  "token": {
    "kind": "ID",
    "lexeme": "abc"
  }
},
{
  "kind": "Terminal",
  "token": {
    "kind": "ASSIGN",
    "lexeme": "="
  }
},
{
  "type": "NonTerminal",
  "kind": "Expr",
  "children": [
    {
      "type": "NonTerminal",
      "kind": "EqualityExpr",
      "children": [
        {
          "type": "NonTerminal",
          "kind": "RelationExpr",
          "children": [
            {
              "type": "NonTerminal",
              "kind": "AddExpr",
              "children": [
                {
                  "type": "NonTerminal",
                  "kind": "MulExpr",
                  "children": [
                    {
                      "type": "NonTerminal",
                      "kind": "UnaryExpr",
                      "children": [
                        {
                          "type": "NonTerminal",
                          "kind": "PowerExpr",
                          "children": [
                            {
                              "type": "NonTerminal",
                              "kind": "PrimaryExpr",
                              "children": [
                                {
                                  "kind": "Terminal",
                                  "token": {
                                    "kind": "ID",
                                    "lexeme": "x"
                                  }
                                }
                              ]
                            }
                          ]
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
},
{
  "type": "NonTerminal",

```



412		"children": []
413		}
414		]
415		}
416		]
417		},
418		{
419		"type": "NonTerminal",
420		"kind": "MulExpr'",
421		"children": []
422		}
423		]
424		},
425		{
426		"type": "NonTerminal",
427		"kind": "AddExpr'",
428		"children": [
429		{
430		"kind": "Terminal",
431		"token": {
432		"kind": "OP1",
433		"lexeme": "+"
434		}
435		},
436		{
437		"type": "NonTerminal",
438		"kind": "MulExpr",
439		"children": [
440		{
441		"type": "NonTerminal",
442		"kind": "UnaryExpr",
443		"children": [
444		{
445		"type":
446	"NonTerminal",	"kind":
447	"PowerExpr",	"children": [
448		{
449		"type":
450	"NonTerminal",	"kind":
451	"PrimaryExpr",	"children": [
452		{
453		"kind":
454	"Terminal",	"token": {
455		"kind":
456	"NUM_FLOAT",	"lexeme":
457	"23.3"	}
458		}
459		]
460		},
461		{

```

462                                     "type":
"NonTerminal",
463                                     "kind":
"PowerExpr'",
464                                     "children": []
465                                 }
466                             ]
467                         }
468                     ]
469                 },
470                 {
471                     "type": "NonTerminal",
472                     "kind": "MulExpr'",
473                     "children": []
474                 }
475             ]
476         },
477         {
478             "type": "NonTerminal",
479             "kind": "AddExpr'",
480             "children": []
481         }
482     ]
483 }
484 ]
485 }
486 ]
487 },
488 {
489     "type": "NonTerminal",
490     "kind": "RelationExpr'",
491     "children": []
492 }
493 ]
494 },
495 {
496     "type": "NonTerminal",
497     "kind": "EqualityExpr'",
498     "children": []
499 }
500 ]
501 }
502 ]
503 },
504 {
505     "kind": "Terminal",
506     "token": {
507         "kind": "SEP",
508         "lexeme": ";"
509     }
510 }
511 ]
512 }
513 ]
514 },
515 {
516     "type": "NonTerminal",
517     "kind": "StmtList",

```

```

518 "children": [
519 {
520   "type": "NonTerminal",
521   "kind": "Stmt",
522   "children": [
523     {
524       "type": "NonTerminal",
525       "kind": "BlockMaybeRepeat",
526       "children": [
527         {
528           "type": "NonTerminal",
529           "kind": "Block",
530           "children": [
531             {
532               "kind": "Terminal",
533               "token": {
534                 "kind": "SEP",
535                 "lexeme": "{"
536               }
537             },
538             {
539               "type": "NonTerminal",
540               "kind": "StmtList",
541               "children": [
542                 {
543                   "type": "NonTerminal",
544                   "kind": "Stmt",
545                   "children": [
546                     {
547                       "type": "NonTerminal",
548                       "kind": "AssignStmt",
549                       "children": [
550                         {
551                           "kind": "Terminal",
552                           "token": {
553                             "kind": "ID",
554                             "lexeme": "abc"
555                           }
556                         },
557                         {
558                           "kind": "Terminal",
559                           "token": {
560                             "kind": "ASSIGN",
561                             "lexeme": "="
562                           }
563                         },
564                         {
565                           "type": "NonTerminal",
566                           "kind": "Expr",
567                           "children": [
568                             {
569                               "type": "NonTerminal",
570                               "kind": "EqualityExpr",
571                               "children": [
572                                 {
573                                   "type": "NonTerminal",
574                                   "kind": "RelationExpr",
575                                   "children": [

```

576		{
577		"type":
578		"kind": "AddExpr",
579		"children": [
580		{
581		"type":
582		"kind":
583		"children": [
584		{
585		"type":
586		"kind":
587		"children":
588		{
589		"type":
590		"kind":
591		
592		
593		
594		"type": "NonTerminal",
595		"kind": "PrimaryExpr",
596		"children": [
597		{
598		"kind": "Terminal",
599		"token": {
600		"kind": "ID",
601		"lexeme": "abc"
602		}
603		}
604		},
605		{
606		
607		"type": "NonTerminal",
608		"kind": "PowerExpr'",
609		"children": []
610		}
611		]
612		}
613		},



614		{
615		"type":
616	"NonTerminal",	
617		"kind":
618	"MulExpr'",	"children":
619	[	{
620		"kind":
621	"Terminal",	
622	"token": {	
623	"kind": "OP1",	}
624	"lexeme": "*"	},
625		{
626		"type":
627	"NonTerminal",	"kind":
628	"UnaryExpr",	
629	"children": [	{
630		
631	"type": "NonTerminal",	
632	"kind": "PowerExpr",	
633	"children": [	{
634		
635	"type": "NonTerminal",	
636	"kind": "PrimaryExpr",	
637	"children": [	
638	{	
639	"kind": "Terminal",	
640	"token": {	
641	"kind": "NUM_FLOAT",	
642	"lexeme": "23.3"	
643	}	
644	}	
645	]	
646	},	{

```

647     "type": "NonTerminal",
648     "kind": "PowerExpr'",
649     "children": []
650         }
651     ]
652 }
653 ]
654 },
655 {
656     "type":
657         "NonTerminal",
658         "kind":
659         "MulExpr'",
660         "children": []
661     }
662 ]
663 },
664 {
665     "type":
666         "NonTerminal",
667         "kind":
668         "AddExpr'",
669         "children": []
670     }
671 ]
672 },
673 {
674     "type":
675         "NonTerminal",
676         "kind":
677         "RelationExpr'",
678         "children": []
679     }
680 ]
681 },
682 {
683     "type": "NonTerminal",
684     "kind":
685         "EqualityExpr'",
686         "children": []
687     }
688 ]
689 },
690 {
691     "kind": "Terminal",
692     "token": {
693         "kind": "SEP",
694         "lexeme": ";"
695     }
696 }

```

```

694         ]
695     }
696 ]
697 },
698 {
699     "type": "NonTerminal",
700     "kind": "StmtList",
701     "children": []
702 }
703 ]
704 },
705 {
706     "kind": "Terminal",
707     "token": {
708         "kind": "SEP",
709         "lexeme": "}"
710     }
711 }
712 ]
713 },
714 {
715     "type": "NonTerminal",
716     "kind": "RepeatTail",
717     "children": [
718         {
719             "kind": "Terminal",
720             "token": {
721                 "kind": "REPEAT",
722                 "lexeme": "^^"
723             }
724         },
725         {
726             "type": "NonTerminal",
727             "kind": "Expr",
728             "children": [
729                 {
730                     "type": "NonTerminal",
731                     "kind": "EqualityExpr",
732                     "children": [
733                         {
734                             "type": "NonTerminal",
735                             "kind": "RelationExpr",
736                             "children": [
737                                 {
738                                     "type": "NonTerminal",
739                                     "kind": "AddExpr",
740                                     "children": [
741                                         {
742                                             "type": "NonTerminal",
743                                             "kind": "MulExpr",
744                                             "children": [
745                                                 {
746                                                     "type": "NonTerminal",
747                                                     "kind": "UnaryExpr",
748                                                     "children": [
749                                                         {
750                     "type":

```

"NonTerminal",

751		"kind":
	"PowerExpr",	
752		"children": [
753		{
754		"type":
	"NonTerminal",	
755		"kind":
	"PrimaryExpr",	
756		"children": [
757		{
758		"kind":
	"Terminal",	
759		"token": {
760		"kind":
	"NUM_INT",	
761		"lexeme":
	"66"	
762		}
763		}
764		]
765		},
766		{
767		"type":
	"NonTerminal",	
768		"kind":
	"PowerExpr'",	
769		"children": []
770		}
771		]
772		}
773		]
774		},
775		{
776		"type": "NonTerminal",
777		"kind": "MulExpr'",
778		"children": []
779		}
780		]
781		},
782		{
783		"type": "NonTerminal",
784		"kind": "AddExpr'",
785		"children": []
786		}
787		]
788		},
789		{
790		"type": "NonTerminal",
791		"kind": "RelationExpr'",
792		"children": []
793		}
794		]
795		},
796		{
797		"type": "NonTerminal",
798		"kind": "EqualityExpr'",
799		"children": []
800		}

```

801         ]
802     }
803 ]
804 },
805 {
806     "kind": "Terminal",
807     "token": {
808         "kind": "SEP",
809         "lexeme": ";"
810     }
811 }
812 ]
813 }
814 ]
815 }
816 ]
817 },
818 {
819     "type": "NonTerminal",
820     "kind": "StmtList",
821     "children": [
822     {
823         "type": "NonTerminal",
824         "kind": "Stmt",
825         "children": [
826         {
827             "type": "NonTerminal",
828             "kind": "IfStmt",
829             "children": [
830             {
831                 "kind": "Terminal",
832                 "token": {
833                     "kind": "KEY",
834                     "lexeme": "if"
835                 }
836             },
837             {
838                 "kind": "Terminal",
839                 "token": {
840                     "kind": "SEP",
841                     "lexeme": "("
842                 }
843             },
844             {
845                 "type": "NonTerminal",
846                 "kind": "Expr",
847                 "children": [
848                 {
849                     "type": "NonTerminal",
850                     "kind": "EqualityExpr",
851                     "children": [
852                     {
853                         "type": "NonTerminal",
854                         "kind": "RelationExpr",
855                         "children": [
856                         {
857                             "type": "NonTerminal",
858                             "kind": "AddExpr",

```

859		"children": [
860		{
861		"type": "NonTerminal",
862		"kind": "MulExpr",
863		"children": [
864		{
865		"type": "NonTerminal",
866		"kind": "UnaryExpr",
867		"children": [
868		{
869		"type":
870	"NonTerminal",	"kind":
871	"PowerExpr",	"children": [
872		{
873		"type":
874	"NonTerminal",	"kind":
875	"PrimaryExpr",	"children": [
876		{
877		"kind":
878	"Terminal",	"token": {
879		"kind":
880	"ID",	"lexeme":
881	"x"	}
882		}
883		]
884		},
885		{
886		"type":
887	"NonTerminal",	"kind":
888	"PowerExpr'",	"children": []
889		}
890		]
891		}
892		]
893		},
894		{
895		"type": "NonTerminal",
896		"kind": "MulExpr'",
897		"children": []
898		}
899		]
900		},
901		{
902		"type": "NonTerminal",
903		"kind": "AddExpr'",
904		"children": []
905		}
906		]
907		},

908		{
909		"type": "NonTerminal",
910		"kind": "RelationExpr",
911		"children": [
912		{
913		"kind": "Terminal",
914		"token": {
915		"kind": "OP2",
916		"lexeme": ">="
917		}
918		},
919		{
920		"type": "NonTerminal",
921		"kind": "AddExpr",
922		"children": [
923		{
924		"type": "NonTerminal",
925		"kind": "MulExpr",
926		"children": [
927		{
928		"type":
929	"NonTerminal",	"kind":
930	"UnaryExpr",	"children": [
931		{
932		"type":
933	"NonTerminal",	"kind":
934	"PowerExpr",	"children": [
935		{
936		"type":
937	"NonTerminal",	"kind":
938	"PrimaryExpr",	"children":
939	[	{
940		"kind":
941	"Terminal",	
942	"token": {	
943	"kind": "ID",	
944	"lexeme": "y"	}
945		}
946		]
947		},
948		{
949	"NonTerminal",	"type":
950	"PowerExpr'",	"kind":
951		"children":
	[ ]	

952		}
953		]
954		}
955		]
956		},
957		{
958		"type":
	"NonTerminal",	
959		"kind": "MulExpr",
960		"children": []
961		}
962		]
963		},
964		{
965		"type": "NonTerminal",
966		"kind": "AddExpr",
967		"children": [
968		{
969		"kind": "Terminal",
970		"token": {
971		"kind": "OP1",
972		"lexeme": "+"
973		}
974		},
975		{
976		"type":
	"NonTerminal",	
977		"kind": "MulExpr",
978		"children": [
979		{
980		"type":
	"NonTerminal",	
981		"kind":
	"UnaryExpr",	
982		"children": [
983		{
984		"type":
	"NonTerminal",	
985		"kind":
	"PowerExpr",	
986		"children":
	[	
987		{
988		"type":
	"NonTerminal",	
989		"kind":
	"PrimaryExpr",	
990		}
	"children": [	
991		{
992		"kind": "Terminal",
993		"token": {
994		"kind": "ID",
995		"lexeme": "x"



```

996                                     }
997                                     }
998                                 ]
999                             },
1000                             {
1001                                 "type":
"NonTerminal",
1002                                 "kind":
"PowerExpr'",
1003                                 "children": []
1004                             }
1005                         ]
1006                     }
1007                 ]
1008             },
1009             {
1010                 "type":
"NonTerminal",
1011                 "kind":
"MulExpr'",
1012                 "children": []
1013             }
1014         ]
1015     },
1016     {
1017         "type":
"NonTerminal",
1018         "kind": "AddExpr'",
1019         "children": []
1020     }
1021 ]
1022 }
1023 ]
1024 },
1025 {
1026     "type": "NonTerminal",
1027     "kind": "RelationExpr'",
1028     "children": []
1029 }
1030 ]
1031 }
1032 ]
1033 },
1034 {
1035     "type": "NonTerminal",
1036     "kind": "EqualityExpr'",
1037     "children": [
1038         {
1039             "kind": "Terminal",
1040             "token": {
1041                 "kind": "OP2",
1042                 "lexeme": "=="
1043             }
1044         },
1045         {
1046             "type": "NonTerminal",
1047             "kind": "RelationExpr",

```

```
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
```

"NonTerminal",

"UnaryExpr",

"NonTerminal",

"PowerExpr",

"NonTerminal",

"PrimaryExpr",

[

"Terminal",

"token": {

"kind": "ID",

"lexeme": "y"

"NonTerminal",

"PowerExpr'",

[]

"NonTerminal",

```
"children": [
{
  "type": "NonTerminal",
  "kind": "AddExpr",
  "children": [
    {
      "type": "NonTerminal",
      "kind": "MulExpr",
      "children": [
        {
          "type":

          "kind":

          "children": [
            {
              "type":

              "kind":

              "children":

              {
                "kind":

              }
            }
          ],
          },
          {
            "type":

            "kind":

            "children":

          }
        ]
      }
    ],
    {
      "type":

      "kind": "MulExpr'",
      "children": []
    }
  ]
}
```

1091		}
1092		]
1093		},
1094		{
1095		"type": "NonTerminal",
1096		"kind": "AddExpr",
1097		"children": [
1098		{
1099		"kind": "Terminal",
1100		"token": {
1101		"kind": "OP1",
1102		"lexeme": "+"
1103		}
1104		},
1105		{
1106		"type":
1107	"NonTerminal",	
1108		"kind": "MulExpr",
1109		"children": [
1110		{
1111	"NonTerminal",	"type":
1112	"UnaryExpr",	"kind":
1113		"children": [
1114		{
1115	"NonTerminal",	"type":
1116	"PowerExpr",	"kind":
1117		"children":
1118	[	{
1119	"NonTerminal",	"type":
1120	"PrimaryExpr",	"kind":
1121	"children": [	{
1122		"kind": "Terminal",
1123	"token": {	
1124	"kind": "ID",	
1125	"lexeme": "x"	
1126		}
1127		}
1128		]
1129		},
1130		{
1131	"NonTerminal",	"type":
1132	"PowerExpr'",	"kind":

1133	"children": []	
1134		}
1135		]
1136		}
1137		]
1138		},
1139		{
1140		"type":
1141	"NonTerminal",	"kind":
1142		"children": []
1143		}
1144		]
1145		},
1146		{
1147		"type":
1148	"NonTerminal",	"kind": "AddExpr",
1149		"children": []
1150		}
1151		]
1152		}
1153		]
1154		},
1155		{
1156		"type": "NonTerminal",
1157		"kind": "RelationExpr",
1158		"children": []
1159		}
1160		]
1161		},
1162		{
1163		"type": "NonTerminal",
1164		"kind": "EqualityExpr",
1165		"children": [
1166		{
1167		"kind": "Terminal",
1168		"token": {
1169		"kind": "OP2",
1170		"lexeme": "!="
1171		}
1172		},
1173		{
1174		"type": "NonTerminal",
1175		"kind": "RelationExpr",
1176		"children": [
1177		{
1178		"type": "NonTerminal",
1179		"kind": "AddExpr",
1180		"children": [
1181		{
1182	"NonTerminal",	"type":
1183		"kind": "MulExpr",
1184		"children": [
1185		{

1186		"type":
1187	"NonTerminal",	"kind":
1188	"UnaryExpr",	"children": [
1189		{
1190	"NonTerminal",	"type":
1191	"PowerExpr",	"kind":
1192		"children":
1193	[	{
1194	"NonTerminal",	"type":
1195	"PrimaryExpr",	"kind":
1196	"children": [	
1197		{
1198	"kind": "Terminal",	
1199	"token": {	
1200	"kind": "ID",	
1201	"lexeme": "y"	
1202		}
1203		}
1204		]
1205		},
1206		{
1207	"NonTerminal",	"type":
1208	"PowerExpr'",	"kind":
1209	"children": []	
1210		}
1211		]
1212		}
1213		]
1214		},
1215		{
1216	"NonTerminal",	"type":
1217	"MulExpr'",	"kind":
1218		"children": []
1219		}
1220		]
1221		},
1222		{
1223	"NonTerminal",	"type":
1224		"kind": "AddExpr'",
1225		"children": []

```

1226         }
1227     ]
1228 },
1229 {
1230     "type": "NonTerminal",
1231     "kind":
"RelationExpr",
1232     "children": []
1233 }
1234 ]
1235 },
1236 {
1237     "type": "NonTerminal",
1238     "kind": "EqualityExpr",
1239     "children": []
1240 }
1241 ]
1242 }
1243 ]
1244 }
1245 ]
1246 }
1247 ]
1248 },
1249 {
1250     "kind": "Terminal",
1251     "token": {
1252         "kind": "SEP",
1253         "lexeme": ")"
1254     }
1255 },
1256 {
1257     "type": "NonTerminal",
1258     "kind": "Block",
1259     "children": [
1260         {
1261             "kind": "Terminal",
1262             "token": {
1263                 "kind": "SEP",
1264                 "lexeme": "{"
1265             }
1266         },
1267         {
1268             "type": "NonTerminal",
1269             "kind": "StmtList",
1270             "children": [
1271                 {
1272                     "type": "NonTerminal",
1273                     "kind": "Stmt",
1274                     "children": [
1275                         {
1276                             "type": "NonTerminal",
1277                             "kind": "FuncCallStmt",
1278                             "children": [
1279                                 {
1280                                     "type": "NonTerminal",
1281                                     "kind": "FuncCall",
1282                                     "children": [

```

1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
  
1303  
  
1304  
1305  
1306  
  
1307  
  
1308  
1309  
1310  
  
1311  
  
1312  
  
1313  
1314  
  
1315  
  
1316  
  
1317  
1318  
  
1319  
  
1320  
  
1321  
1322  
  
1323  
  
1324

```
"NonTerminal",  
"EqualityExpr",  
  
"NonTerminal",  
"RelationExpr",  
  
"NonTerminal",  
"AddExpr",  
[  
"NonTerminal",  
"MulExpr",  
  "children": [  
    "type": "NonTerminal",  
    "kind": "UnaryExpr",  
    "children": [  
      "type": "NonTerminal",  
      "kind": "PowerExpr",  
      "children": [  

```

```
{  
  "kind": "Terminal",  
  "token": {  
    "kind": "KEY",  
    "lexeme": "print"  
  }  
},  
{  
  "kind": "Terminal",  
  "token": {  
    "kind": "SEP",  
    "lexeme": "("  
  }  
},  
{  
  "type": "NonTerminal",  
  "kind": "Expr",  
  "children": [  
    {  
      "type":  
  
      "kind":  
  
      "children": [  
        {  
          "type":  
  
          "kind":  
  
          "children":  
  
            {  
              "type":  
  
              "kind":  
  
            }  
          }  
        }  
      }  
    }  
  ]  
}
```

1325	{	
1326	"type": "NonTerminal",	
1327	"kind": "PrimaryExpr",	
1328	"children": [	
1329	{	
1330	"kind": "Terminal",	
1331	"token": {	
1332	"kind": "ID",	
1333	"lexeme": "x"	
1334	}	
1335	}	
1336	]	
1337	},	
1338	{	
1339	"type": "NonTerminal",	
1340	"kind": "PowerExpr'",	
1341	"children": []	
1342	}	
1343	]	
1344		}
1345		]
1346		},
1347		{
1348	"type": "NonTerminal",	
1349	"kind": "MulExpr'",	
1350	"children": []	
1351		}
1352		]
1353		},
1354		{
1355	"type":	
1356	"NonTerminal",	
1357	"kind":	
	"AddExpr'",	
	"children": []	



1358		}
1359		]
1360		},
1361		{
1362		"type":
1363	"NonTerminal",	"kind":
1364		"children":
1365		}
1366		]
1367		},
1368		{
1369		"type":
1370	"NonTerminal",	"kind":
1371		"children": []
1372		}
1373		]
1374		}
1375		]
1376		},
1377		{
1378		"kind": "Terminal",
1379		"token": {
1380		"kind": "SEP",
1381		"lexeme": ")"
1382		}
1383		}
1384		]
1385		},
1386		{
1387		"kind": "Terminal",
1388		"token": {
1389		"kind": "SEP",
1390		"lexeme": ";"
1391		}
1392		}
1393		]
1394		}
1395		]
1396		},
1397		{
1398		"type": "NonTerminal",
1399		"kind": "StmtList",
1400		"children": []
1401		}
1402		]
1403		},
1404		{
1405		"kind": "Terminal",
1406		"token": {
1407		"kind": "SEP",
1408		"lexeme": "}"
1409		}
1410		}

1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467

"NonTerminal",

```
]
},
{
  "type": "NonTerminal",
  "kind": "ElsePart",
  "children": [
    {
      "kind": "Terminal",
      "token": {
        "kind": "KEY",
        "lexeme": "else"
      }
    }
  ],
  {
    "type": "NonTerminal",
    "kind": "Block",
    "children": [
      {
        "kind": "Terminal",
        "token": {
          "kind": "SEP",
          "lexeme": "{"
        }
      }
    ],
    {
      "type": "NonTerminal",
      "kind": "StmtList",
      "children": [
        {
          "type": "NonTerminal",
          "kind": "Stmt",
          "children": [
            {
              "type": "NonTerminal",
              "kind": "FuncCallStmt",
              "children": [
                {
                  "type": "NonTerminal",
                  "kind": "FuncCall",
                  "children": [
                    {
                      "kind": "Terminal",
                      "token": {
                        "kind": "KEY",
                        "lexeme": "print"
                      }
                    }
                  ],
                },
                {
                  "kind": "Terminal",
                  "token": {
                    "kind": "SEP",
                    "lexeme": "("
                  }
                }
              ],
            },
            {
              "type":
                "kind": "Expr",
```

```

1468                                     "children": [
1469                                         {
1470                                             "type":
1471                                                 "NonTerminal",
1472                                             "kind":
1473                                                 "EqualityExpr",
1474                                             "children": [
1475                                                 {
1476                                                     "type":
1477                                                         "NonTerminal",
1478                                                     "kind":
1479                                                         "RelationExpr",
1480                                                     "children": [
1481                                                         {
1482                                                             "type": "NonTerminal",
1483                                                             "kind": "MulExpr",
1484                                                             "children": [
1485                                                                 {
1486                                                                     "type": "NonTerminal",
1487                                                                     "kind": "UnaryExpr",
1488                                                                     "children": [
1489                                                                         {
1490                                                                             "type": "NonTerminal",
1491                                                                             "kind": "PowerExpr",
1492                                                                             "children": [
1493                                                                                 {
1494                                                                                     "type": "NonTerminal",
1495                                                                                     "kind": "PrimaryExpr",
1496                                                                                     "children": [
1497                                                                                         {
1498                             "kind": "Terminal",
1499                             "token": {

```

1500		
	"kind": "ID",	
1501		
	"lexeme": "y"	
1502		
	}	
1503		
	}	
1504		
	]	
1505		
	},	
1506		
	{	
1507		
	"type": "NonTerminal",	
1508		
	"kind": "PowerExpr'",	
1509		
	"children": []	
1510		
	}	
1511		
	]	
1512		
	}	
1513		
	]	
1514		
	},	
1515		{
1516		
	"type": "NonTerminal",	
1517		
	"kind": "MulExpr'",	
1518		
	"children": []	
1519		}
1520		]
1521		},
1522		{
1523		
	"type": "NonTerminal",	
1524		
	"kind": "AddExpr'",	
1525		
	"children": []	
1526		}
1527		]
1528		},
1529		{
1530		"type":
	"NonTerminal",	
1531		"kind":
	"RelationExpr'",	
1532		
	"children": []	
1533		}

```

1534                                     ]
1535                                     },
1536                                     {
1537                                         "type":
"NonTerminal",
1538                                         "kind":
"EqualityExpr'",
1539                                         "children":
[]
1540                                     }
1541                                 ]
1542                             }
1543                         ]
1544                     },
1545                     {
1546                         "kind": "Terminal",
1547                         "token": {
1548                             "kind": "SEP",
1549                             "lexeme": ")"
1550                         }
1551                     }
1552                 ]
1553             },
1554             {
1555                 "kind": "Terminal",
1556                 "token": {
1557                     "kind": "SEP",
1558                     "lexeme": ";"
1559                 }
1560             }
1561         ]
1562     }
1563 ]
1564 },
1565 {
1566     "type": "NonTerminal",
1567     "kind": "StmtList",
1568     "children": []
1569 }
1570 ]
1571 },
1572 {
1573     "kind": "Terminal",
1574     "token": {
1575         "kind": "SEP",
1576         "lexeme": "}"
1577     }
1578 }
1579 ]
1580 }
1581 ]
1582 }
1583 ]
1584 }
1585 ]
1586 },
1587 {
1588     "type": "NonTerminal",

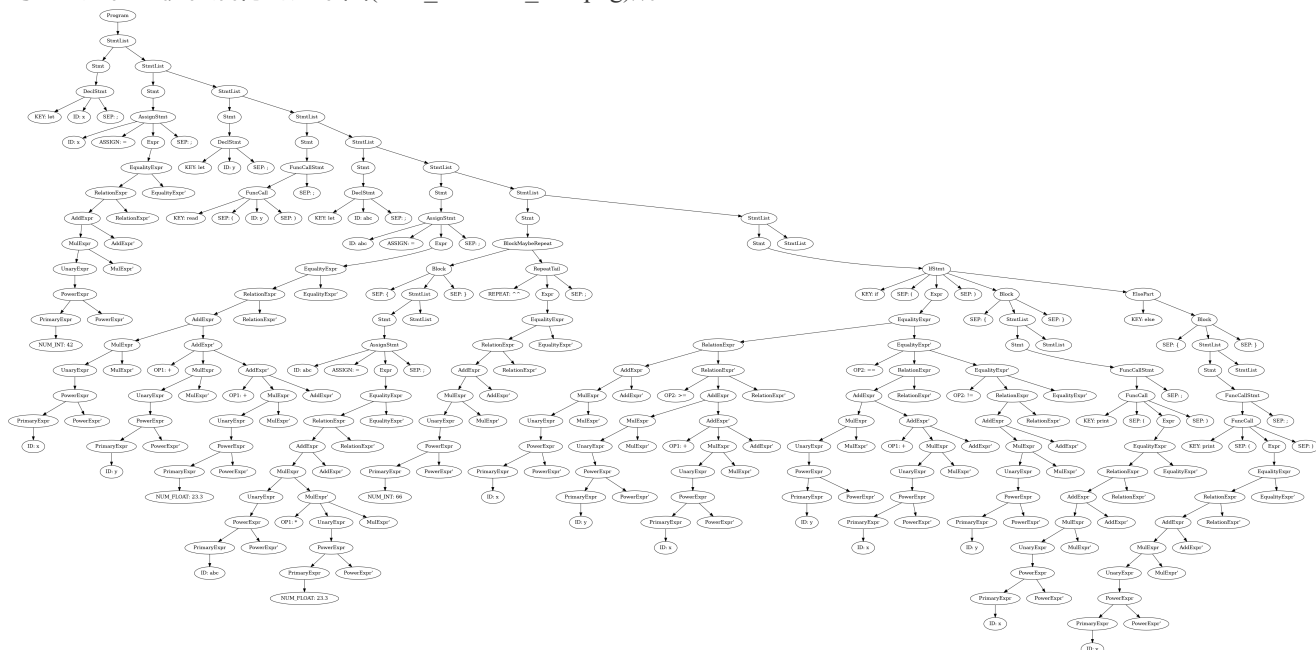
```

```

1589         "kind": "StmtList",
1590         "children": []
1591     }
1592 ]
1593 }
1594 ]
1595 }
1596 ]
1597 }
1598 ]
1599 }
1600 ]
1601 }
1602 ]
1603 }
1604 ]
1605 }
1606 ]
1607 }
1608 ]
1609 }

```

对AST进行可视化后得到如下图(./dfa\_lexer/ast\_tree.png)所示：



## i. 遇到的问题及解决方案

### 词法分析器的“贪婪匹配”失效

现象：输入的运算符 `==` 被错误地识别为两个 `ASSIGN (=)` Token，导致后续语法分析报错。原因：在 `lex_code` 的早期实现中，可能一旦匹配到 `accept` 状态就立即返回了，没有继续尝试匹配更长的字符串。解决方法：

逻辑优化：代码中引入了 `last_accept_state` 和 `last_accept_pos` 变量。

具体做法：即使当前进入了接受状态，循环也不会停止，而是继续读下一个字符，直到无法转移为止。只有当彻底无法匹配时，才回退到“最后一次成功的接受状态”。这就是最大匹配原则（Longest Match）。

## LL(1) 递归构造中的“悬挂子节点”

现象：生成的 AST 结构是扁平的，或者子节点没有正确连接到父节点，导致 JSON 输出只有根节点。原因：在非递归解析中，由于产生式是不断弹栈、压栈的，很难在符号弹出时找到它属于哪个父节点。解决方法：

同步对象设计：在代码中，stack 存储的不只是 Token，而是 (Token, Rc<RefCell> )。

预先绑定：当非终结符展开产生式时，立即创建所有子节点的 Rc 引用，并将其放入父节点的 children 列表中。随后将这些子节点连同符号名一起压入栈。

指针共享：通过 Rc 指针，栈中的任务和树中的节点指向同一个内存地址。当栈处理完终结符并填充 lexeme 时，树结构中的对应节点也同步更新了。

## LL(1) 查表时的“种类失配”

现象：程序报错 No row for non-terminal 'ID'，但 ID 本身应该是终结符。原因：在 parse\_ll1 中，程序有时混淆了“Token 的具体内容”和“Token 的类型”。例如，输入是 NUM(3.14)，但预测表（Table）的列名是 NUM。解决方法：

Key 规范化：引入 Token::to\_key() 方法。

具体实现：

```
1 match self.kind.as_str() {
2     "ID" | "NUM_INT" | "NUM_FLOAT" => Token::from(self.kind.as_str()),
3     _ => self.clone(),
4 }
```

在查表前强制转换，确保对于所有具体的数字，都使用通用的 "NUM" 作为索引去查表。

## JSON 序列化时的“循环引用”或“递归过深”

现象：使用 serde\_json 序列化 AST 时程序崩溃，或生成的 JSON 包含大量重复数据。原因：ASTNode 使用了 Rc<RefCell<...>>，如果处理不当，可能会导致序列化器陷入无限死循环。解决方法：

手动实现 Serialize：代码中通过 impl Serialize for ASTNode 手动控制了序列化行为。

克隆解构：在序列化 NonTerminal 时，通过 .map(|child| child.borrow().clone()) 将引用转换为独立副本，打破了引用计数的复杂链路，确保生成的是标准的树状 JSON。

# j. 心得体会与建议

## 对“数据驱动”编程范式的深刻理解

在实验初期，我倾向于用大量的 if-else 或 match 语句去手写逻辑，但很快发现这种方式在面对复杂语法时难以维护。

感悟：本实验最核心的价值在于通用性。通过加载 min\_dfa.json 和 LL1table.json，程序逻辑与具体的语言语法脱钩了。

收获：我学会了如何设计一个“引擎”，让它能够根据不同的规则配置自动处理不同的编程语言，这种高度抽象的设计思想比单纯的代码实现更为重要。

## Rust 所有权机制与树形结构的博弈

在构建 AST（抽象语法树）时，Rust 的严格所有权系统曾让我感到非常棘手，特别是在非递归的 LL(1) 算法中。

技术突破：为了在栈（Stack）和树（Tree）之间共享节点，我被迫深入学习了 Rc<RefCell> 组合。

心得：理解了如何利用“内部可变性”在保证内存安全的前提下，构建动态增长的复杂数据结构。这不仅解决了 AST 的构建问题，也让我对 Rust 内存管理的底层逻辑有了更真实的体感。

## 理论与实践的“破壁”

以前在书本上学习编译原理时，LL(1) 预测分析表和 DFA 状态转移图只是纸上的推演。

感悟：

词法阶段：亲手实现了“最大匹配原则”后，才真正理解为什么 `==` 不会被识别为两个 `=`。

语法阶段：当看到一串扁平的 Token 序列，通过栈的压入弹出，最终神奇地变成一棵层次分明的 JSON 语法树时，那种“将混乱转化为秩序”的成就是无可比拟的。

反馈：这种“实现一个小型编译器前端”的实验设计非常优秀，它将抽象的代数理论转化为了看得见、摸得着的工程实践。

## 对程序严谨性的敬畏

在实验中，哪怕是正则表达式中少了一个边界符 `\b`，或者是 LL(1) 表中漏掉了一个  $\epsilon$  产生式，整个解析器都会瞬间崩塌。

收获：这培养了我极其严谨的思维习惯。在处理边界情况（如文件末尾 EOF、空行、非法字符）时，必须做到逻辑闭环。这种对细节的把控力是作为一名高级软件工程师必备的素质。



# 课程反馈与建议

如果能在后续加入 错误恢复的内容会更好。目前的程序在遇到语法错误时会直接报错退出，如果能学习如何跳过错误 Token 继续解析，将更贴近真实的生产环境。

## k. 附录

- 输入文件路径：
  - 词法分析器配置文件: `./min_dfa.json`
  - 语法分析器预测表: `./LL1table.json`
  - 测试源代码: `./test_code.txt`
- 输出文件路径：
  - 词法分析器输出 Token 序列: `./dfa_lexer/token.txt`
  - 语法分析器输出 AST JSON: `./dfa_lexer/ast.json`
- 程序源代码路径: `./dfa_lexer/src/`
- 实验报告路径: `./report.md`
- 其他相关文件：
  - AST可视化脚本: `./draw_tree.py`
  - AST可视化结果: `./dfa_lexer/ast_tree.png`













