# Syntax Analyzer 实验报告

151070063 宋欣建

**Motivation/Aim**

自行定义文法，运用语法分析方法对输入语句进行语法分析并输出结果，加深对语

法分析过程的理解。

**Content description**

此程序用Java 编写。程序读取一个文本文件，运用上次实验写的词法分析程序得

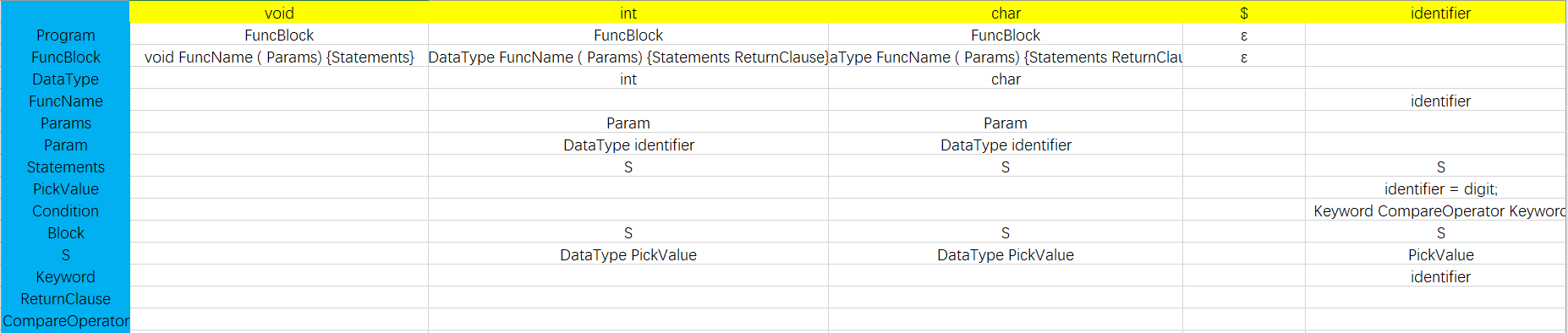
到TOKEN 序列，再对其进行语法分析。这里使用的是LL(1)方法进行自顶向下分析，最

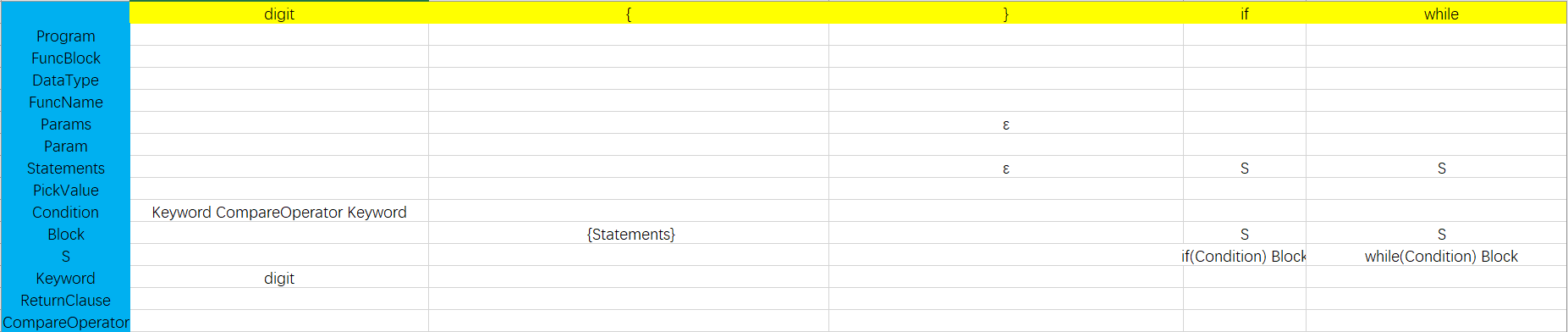
后输出产生式序列。

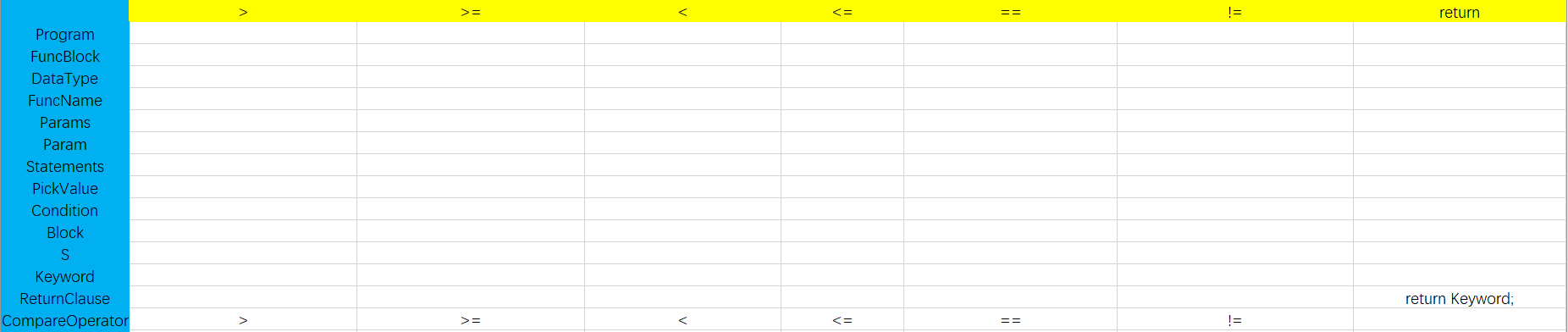
定义的具体文法有：

1. Program → FuncBlock
2. FuncBlock → void FuncName ( Paras ) { Statements }
3. FuncBlock → DataType FuncName ( Paras ) { Statements ReturnClause }
4. DataType → int | char
5. FuncName → id
6. Params →Param
7. Param → DataType id
8. Statements → S
9. S → DataType PickValue
10. S → PickValue
11. PickValue → id = digit ;
12. S → if ( Condition ) Block
13. Block → S
14. Condition → Keyword CompareOperator Keyword
15. Keyword → id | digit
16. CompareOperator → > | >= | < | <= | == | !=
17. S → while ( Condition ) Block
18. ReturnClause → return Keyword ;

计算后得到的PPT为







## Ideas/Methods

* 读入CFG
* 初始化PPT
* 遍历每个产生式，求右侧的First，若为ε，则求左侧的Follow
* 将结果填到PPT中
* 以上均是手工完成，直接把得到的PPT填入代码即可
* 初始化分析栈，底部填入$和开始符
* 处理输入串，末端添加$。遍历输入串，根据当前字符和分析栈顶，查询PPT，并将查得产生式右侧代替栈中的左侧（从右向左依次入栈），
* 输出查得的产生式，直至两个$匹配。
* 收集所有用到的产生式，输出推导序列
* 结束

**Description of important Data Structures**

状态：

STATE{

DONE,INVAR,INADD,INMINUS,INLESS,INMORE,INEQUAL,INEXCLAMATORY,INAND,INDIGIT\_DOT,INOR,INSOLIDUS,INDIGIT,INDECIMALS,ANNOTATION\_ONE\_LINE,ANNOTATION\_MULTI\_LINE,ANNOTATION\_MULTI\_LINE\_ASTERISK,SINGLE\_QUOTE\_MARK,DOUBLE\_QUOTE\_MARK

}

记号：

TOKEN{

KEYWORDS,IDENTIFIER,OPERATOR,DELIMITER,INT,DOUBLE,ANNOTATION

}

记号和符号表的结构：

HashMap<TOKEN,String> map=new HashMap<TOKEN,String>();

initMap() {

map.put(TOKEN.KEYWORDS, "关键字");

map.put(TOKEN.IDENTIFIER, "标识符");

map.put(TOKEN.OPERATOR, "操作符");

map.put(TOKEN.DELIMITER, "分隔符");

map.put(TOKEN.INT, "整数");

map.put(TOKEN.DOUBLE, "浮点数");

map.put(TOKEN.ANNOTATION, "注释符");

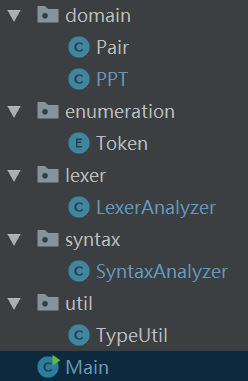
}

关键字数组结构：

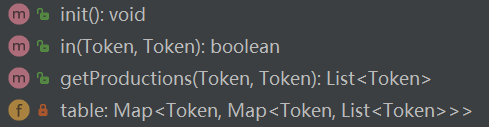
String KEYWORDS[]={"public","class","static","void",

"main","while","if","else","for","switch","case","package","String","int","float","double","char"};

**Description of core Algorithms**

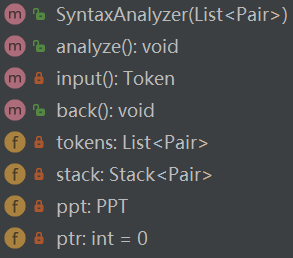


PPT类中把计算后得到的PPT放入进去了



Map<Token,Map<Token,List<Token>>是一个嵌套的Map，外层键是非终结符，内层键是输入字符，查得是一个产生式的右侧，将其拆为一个Token的List。

SyntaxAnalyzer接收LexerAnalyzer输出的一个Token的List。



tokens存放输入的Token序列，ptr是读头指针，指向当前Token，input()返回的是当前Token。

back()是将读头后移的方法。

stack是分析栈。

analyze()是主要分析方法，在循环中比较栈顶和读头下的符号是否一致。

若一致，则弹栈，读头后移；

若不一致，则查PPT，弹栈并将查得的产生式压入栈中。

直至栈顶是结束符$。

**Use cases on running**

输入程序：input.txt

int main(int x) {  
 if (x > 23) {  
 while (5 != 3){  
 x = 200;  
 }  
 }  
 return x;  
}

**输出结果：**

步骤:0

替换!productions:END->[FUNC\_BLOCK]

stack:[(END,$), (FUNC\_BLOCK,FuncBlock)]

步骤:1

替换!productions:END->[DATA\_TYPE, FUNC\_NAME, LEFT\_PARENTHESES, PARAMS, RIGHT\_PARENTHESES, LEFT\_BRACE, STATEMENTS, RETURN\_CLAUSE, RIGHT\_BRACE]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements), (LEFT\_BRACE,{), (RIGHT\_PARENTHESES,)), (PARAMS,Params), (LEFT\_PARENTHESES,(), (FUNC\_NAME,FuncName), (DATA\_TYPE,DataType)]

步骤:2

替换!productions:FUNC\_NAME->[INT]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements), (LEFT\_BRACE,{), (RIGHT\_PARENTHESES,)), (PARAMS,Params), (LEFT\_PARENTHESES,(), (FUNC\_NAME,FuncName), (INT,int)]

步骤:3

匹配! Token为INT

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements), (LEFT\_BRACE,{), (RIGHT\_PARENTHESES,)), (PARAMS,Params), (LEFT\_PARENTHESES,(), (FUNC\_NAME,FuncName)]

步骤:4

替换!productions:LEFT\_PARENTHESES->[IDENTIFIER]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements), (LEFT\_BRACE,{), (RIGHT\_PARENTHESES,)), (PARAMS,Params), (LEFT\_PARENTHESES,(), (IDENTIFIER,identifier)]

步骤:5

匹配! Token为IDENTIFIER

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements), (LEFT\_BRACE,{), (RIGHT\_PARENTHESES,)), (PARAMS,Params), (LEFT\_PARENTHESES,()]

步骤:6

匹配! Token为LEFT\_PARENTHESES

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements), (LEFT\_BRACE,{), (RIGHT\_PARENTHESES,)), (PARAMS,Params)]

步骤:7

替换!productions:RIGHT\_PARENTHESES->[PARAM]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements), (LEFT\_BRACE,{), (RIGHT\_PARENTHESES,)), (PARAM,Param)]

步骤:8

替换!productions:RIGHT\_PARENTHESES->[DATA\_TYPE, IDENTIFIER]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements), (LEFT\_BRACE,{), (RIGHT\_PARENTHESES,)), (IDENTIFIER,identifier), (DATA\_TYPE,DataType)]

步骤:9

替换!productions:IDENTIFIER->[INT]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements), (LEFT\_BRACE,{), (RIGHT\_PARENTHESES,)), (IDENTIFIER,identifier), (INT,int)]

步骤:10

匹配! Token为INT

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements), (LEFT\_BRACE,{), (RIGHT\_PARENTHESES,)), (IDENTIFIER,identifier)]

步骤:11

匹配! Token为IDENTIFIER

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements), (LEFT\_BRACE,{), (RIGHT\_PARENTHESES,))]

步骤:12

匹配! Token为RIGHT\_PARENTHESES

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements), (LEFT\_BRACE,{)]

步骤:13

匹配! Token为LEFT\_BRACE

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (STATEMENTS,Statements)]

步骤:14

替换!productions:RETURN\_CLAUSE->[S]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (S,S)]

步骤:15

替换!productions:RETURN\_CLAUSE->[IF, LEFT\_PARENTHESES, CONDITION, RIGHT\_PARENTHESES, BLOCK]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (CONDITION,Condition), (LEFT\_PARENTHESES,(), (IF,if)]

步骤:16

匹配! Token为IF

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (CONDITION,Condition), (LEFT\_PARENTHESES,()]

步骤:17

匹配! Token为LEFT\_PARENTHESES

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (CONDITION,Condition)]

步骤:18

替换!productions:RIGHT\_PARENTHESES->[KEYWORD, COMPARE\_OPERATOR, KEYWORD]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (KEYWORD,Keyword), (COMPARE\_OPERATOR,CompareOperator), (KEYWORD,Keyword)]

步骤:19

替换!productions:COMPARE\_OPERATOR->[IDENTIFIER]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (KEYWORD,Keyword), (COMPARE\_OPERATOR,CompareOperator), (IDENTIFIER,identifier)]

步骤:20

匹配! Token为IDENTIFIER

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (KEYWORD,Keyword), (COMPARE\_OPERATOR,CompareOperator)]

步骤:21

替换!productions:KEYWORD->[GT]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (KEYWORD,Keyword), (GT,>)]

步骤:22

匹配! Token为GT

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (KEYWORD,Keyword)]

步骤:23

替换!productions:RIGHT\_PARENTHESES->[DIGIT]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (DIGIT,digit)]

步骤:24

匹配! Token为DIGIT

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (BLOCK,Block), (RIGHT\_PARENTHESES,))]

步骤:25

匹配! Token为RIGHT\_PARENTHESES

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (BLOCK,Block)]

步骤:26

替换!productions:RETURN\_CLAUSE->[LEFT\_BRACE, STATEMENTS, RIGHT\_BRACE]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (STATEMENTS,Statements), (LEFT\_BRACE,{)]

步骤:27

匹配! Token为LEFT\_BRACE

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (STATEMENTS,Statements)]

步骤:28

替换!productions:RIGHT\_BRACE->[S]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (S,S)]

步骤:29

替换!productions:RIGHT\_BRACE->[WHILE, LEFT\_PARENTHESES, CONDITION, RIGHT\_PARENTHESES, BLOCK]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (CONDITION,Condition), (LEFT\_PARENTHESES,(), (WHILE,while)]

步骤:30

匹配! Token为WHILE

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (CONDITION,Condition), (LEFT\_PARENTHESES,()]

步骤:31

匹配! Token为LEFT\_PARENTHESES

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (CONDITION,Condition)]

步骤:32

替换!productions:RIGHT\_PARENTHESES->[KEYWORD, COMPARE\_OPERATOR, KEYWORD]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (KEYWORD,Keyword), (COMPARE\_OPERATOR,CompareOperator), (KEYWORD,Keyword)]

步骤:33

替换!productions:COMPARE\_OPERATOR->[DIGIT]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (KEYWORD,Keyword), (COMPARE\_OPERATOR,CompareOperator), (DIGIT,digit)]

步骤:34

匹配! Token为DIGIT

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (KEYWORD,Keyword), (COMPARE\_OPERATOR,CompareOperator)]

步骤:35

替换!productions:KEYWORD->[NOT\_EQ]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (KEYWORD,Keyword), (NOT\_EQ,!=)]

步骤:36

匹配! Token为NOT\_EQ

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (KEYWORD,Keyword)]

步骤:37

替换!productions:RIGHT\_PARENTHESES->[DIGIT]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (BLOCK,Block), (RIGHT\_PARENTHESES,)), (DIGIT,digit)]

步骤:38

匹配! Token为DIGIT

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (BLOCK,Block), (RIGHT\_PARENTHESES,))]

步骤:39

匹配! Token为RIGHT\_PARENTHESES

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (BLOCK,Block)]

步骤:40

替换!productions:RIGHT\_BRACE->[LEFT\_BRACE, STATEMENTS, RIGHT\_BRACE]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (RIGHT\_BRACE,}), (STATEMENTS,Statements), (LEFT\_BRACE,{)]

步骤:41

匹配! Token为LEFT\_BRACE

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (RIGHT\_BRACE,}), (STATEMENTS,Statements)]

步骤:42

替换!productions:RIGHT\_BRACE->[S]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (RIGHT\_BRACE,}), (S,S)]

步骤:43

替换!productions:RIGHT\_BRACE->[PICK\_VALUE]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (RIGHT\_BRACE,}), (PICK\_VALUE,PickValue)]

步骤:44

替换!productions:RIGHT\_BRACE->[IDENTIFIER, EQ, DIGIT, SEMICOLON]

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (RIGHT\_BRACE,}), (SEMICOLON,;), (DIGIT,digit), (EQ,=), (IDENTIFIER,identifier)]

步骤:45

匹配! Token为IDENTIFIER

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (RIGHT\_BRACE,}), (SEMICOLON,;), (DIGIT,digit), (EQ,=)]

步骤:46

匹配! Token为EQ

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (RIGHT\_BRACE,}), (SEMICOLON,;), (DIGIT,digit)]

步骤:47

匹配! Token为DIGIT

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (RIGHT\_BRACE,}), (SEMICOLON,;)]

步骤:48

匹配! Token为SEMICOLON

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,}), (RIGHT\_BRACE,})]

步骤:49

匹配! Token为RIGHT\_BRACE

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause), (RIGHT\_BRACE,})]

步骤:50

匹配! Token为RIGHT\_BRACE

stack:[(END,$), (RIGHT\_BRACE,}), (RETURN\_CLAUSE,ReturnClause)]

步骤:51

替换!productions:RIGHT\_BRACE->[RETURN, KEYWORD, SEMICOLON]

stack:[(END,$), (RIGHT\_BRACE,}), (SEMICOLON,;), (KEYWORD,Keyword), (RETURN,return)]

步骤:52

匹配! Token为RETURN

stack:[(END,$), (RIGHT\_BRACE,}), (SEMICOLON,;), (KEYWORD,Keyword)]

步骤:53

替换!productions:SEMICOLON->[IDENTIFIER]

stack:[(END,$), (RIGHT\_BRACE,}), (SEMICOLON,;), (IDENTIFIER,identifier)]

步骤:54

匹配! Token为IDENTIFIER

stack:[(END,$), (RIGHT\_BRACE,}), (SEMICOLON,;)]

步骤:55

匹配! Token为SEMICOLON

stack:[(END,$), (RIGHT\_BRACE,})]

步骤:56

匹配! Token为RIGHT\_BRACE

stack:[(END,$)]

结束!