

## 实验二 递归下降分析器的设计与实现

### 一. 实验目的

了解和掌握递归下降分析方法，编程实现递归下降分析器，即按照给定的文法规则对输入的符号串（词法分析中生成的\*.dyd 文件）是否构成了合法的句子或程序作出判断，并对发现的语法错误给出提示信息。

### 二. 实验内容

#### 1. 变量名表

变量名 vname: char(16)

所属过程 vproc:char(16)

分类 vkind: 0..1(0—变量、1—形参)

变量类型 vtype: types

变量层次 vlev: int

变量在变量表中的位置 vadr: int(相对第一个变量而言)

types=(ints)

#### 2. 过程名表

过程名 pname: char(16)

过程类型 ptype: types

过程层次 plev: int

第一个变量在变量表中的位置 fadr: int

最后一个变量在变量表中的位置 ladr: int

#### 3. 实现提示

1)（有过程说明时）设一个总的变量名表，查、填表时注意嵌套。

2) 语法错分类:

(1)缺少符号错;

(2)符号匹配错;

(3)符号无定义或重复定义。

3) 递归下降分析时,必须先消除左递归。

4. 消除左递归例:

G(E)	$E \rightarrow E+T \mid T$
	$T \rightarrow T * F \mid F$
	$F \rightarrow (E) \mid i$
消除左递归:	$E \rightarrow T \mid E'$
	$E' \rightarrow +TE' \mid \varepsilon$
	$T \rightarrow FT'$
	$T' \rightarrow *FT' \mid \varepsilon$
	$F \rightarrow (E) \mid i$

### 三、实验要求

实验前要做好充分准备,包括程序清单、调试步骤、调试方法。

实验后要进行对程序结果的详细分析等。

### 四. 实验过程及结果

#### 1. 设计思想

语法分析器是编译程序中重要的一个环节,它按照文法对词法分析器输出的二元式序列进行语法分析,建立符号名表和过程名表,将二元式序列继续交给下面的语义分析器进行语义分析。在语法分析的过程中,可能会遇到诸如缺少符号错、符号匹配错、符号无定义、符号重复定义等多种错误,这就同时要求了错误的处理。

本实验语法分析器的设计思想如下图所示:(本实验采用 Java 语言)

首先,对词法分析器输出的二元式序列进行预处理,存放在两个 ArrayList 中,其中一个存放单词,一个存放种别。

然后,开始进行语法分析,从状态 A 开始,进行递归下降语法分析,具体语法分析算法请参见总控算法。语法分析过程中产生的变量和过程,本人统一暂存

在两个 ArrayList 中，在递归下降语法分析结束之后，统一写入变量名表 variables.var 和过程名表 procedures.pro。如果语法分析未分析出语法错误，则将二元式序列完整地写入 result.dys 文件中。如果语法分析分析出语法错误，则在 result.dys 文件中写入 “Unable to pass the grammatical analysis”，表示语法分析未通过，并且将错误写入 error.err 文件中。

对于出错处理，存在 4 种错误：缺少符号错、符号匹配错、符号无定义、符号重复定义。

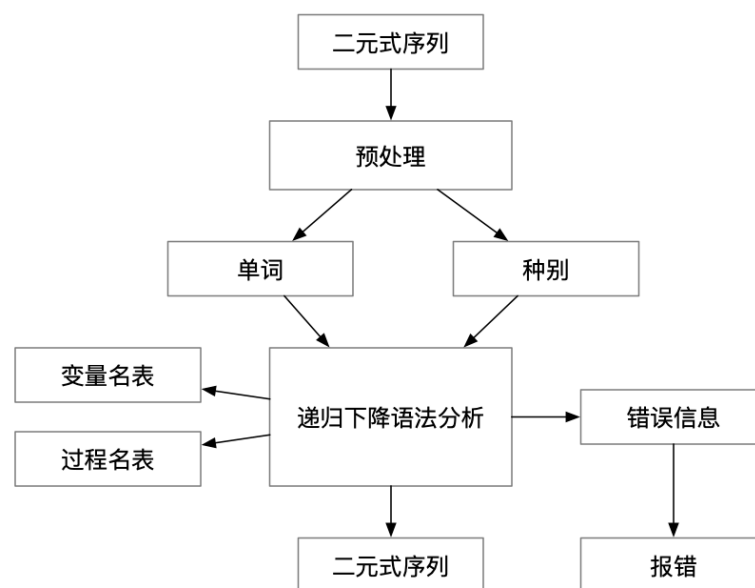


图 1. 语法分析器设计思想

## 2. 总控算法

本人设计的语法分析器中递归下降语法分析算法分析如下：

首先，本人对实验所给的文法进行分析，得到文法产生式。

然后，便可以根据文法产生式编写递归下降的算法。比方说对于文法  $D \rightarrow E|J$ ，就是在函数  $D()$  的函数体内调用函数  $E()$  和  $J()$ 。

本人编写的算法中有两个注意点。首先，因为在词法分析器中已经对标识符的合法性进行了判断，所以在代表标识符的函数  $G()$  中，本人直接对单词的种别进行判断，如果是标识符，则直接分析下一个单词，代表常数的函数  $U()$  同理。其次，本人对于每个错误的处理方式不同。对于缺少符号错，符号无定义和符号重复定义错误，本人报完错之后便根据下一个单词进行下个单词的分析。对于符号匹配错，本人则报完错之后直接跳到下一行，具体见调试过程。

A: 程序	$A \rightarrow B$
B: 分程序	$B \rightarrow \text{begin } C; M \text{ end}$
C: 说明语句表	$C \rightarrow DC'$ $C' \rightarrow ; DC' \mid \varepsilon$
D: 说明语句	$D \rightarrow E \mid J$
E: 变量说明	$E \rightarrow \text{integer } F$
F: 变量	$F \rightarrow G$
G: 标识符	$G \rightarrow HG'$ $G' \rightarrow HG' \mid IG' \mid \varepsilon$
H: 字母	$H \rightarrow a \mid \dots \mid z \mid A \mid \dots \mid Z$
I: 数字	$I \rightarrow 0 \mid 1 \mid \dots \mid 9$
J: 函数说明	$J \rightarrow \text{integer function } G(K); L$
K: 参数	$K \rightarrow F$
L: 函数体	$L \rightarrow \text{begin } C; M \text{ end}$
M: 执行语句表	$M \rightarrow NM'$ $M' \rightarrow ; NM' \mid \varepsilon$
N: 执行语句	$N \rightarrow O \mid P \mid Q \mid W$
O: 读语句	$O \rightarrow \text{read}(F)$
P: 写语句	$P \rightarrow \text{write}(F)$
Q: 赋值语句	$Q \rightarrow F := R$
R: 算术表达式	$R \rightarrow SR'$ $R' \rightarrow -SR' \mid \varepsilon$
S: 项	$S \rightarrow TS'$ $S' \rightarrow *TS' \mid \varepsilon$
T: 因子	$T \rightarrow F \mid U \mid Z$
U: 常数	$U \rightarrow V$
V: 无符号整数	$V \rightarrow IV'$ $V' \rightarrow IV' \mid \varepsilon$
W: 条件语句	$W \rightarrow \text{if } X \text{ then } N \text{ else } N$
X: 条件表达式	$X \rightarrow RYR$
Y: 关系运算符	$Y \rightarrow < \mid < = \mid > \mid > = \mid = \mid < >$
Z: 函数调用	$Z \rightarrow G(R)$

图 2. 文法产生式

### 3. 主要服务子程序算法

本人设计的递归下降语法分析器的最主要的服务子程序算法是预处理算法。

```
start: file                // 存放二元式序列的文件
      input                // 存放二元式单词的 ArrayList
      type                 // 存放二元式种别的 ArrayList
      str                  // 存放二元式序列文件的一行的字符串
begin
      substring1           // 存放 str 的单词部分
      substring 2         // 存放 str 的种别部分
      while str = file.readLine isNot null do
          substring1 = str.substring(0,16)
          substring1 = substring1.trim()    // 去除空格
          input.add(substring1)            // 存入单词表
          substring2 = str.substring(17)
          type.add(substring2)             // 存入种别表
      end
end
```

### 4. 测试程序及运行结果

需要说明的是，词法分析中所给的测试程序存在了符号未定义错误。

首先测试的是没有语法错误的标准测试程序：

```
begin
integer k;
integer m;
integer function F(n);
begin
integer n;
if n<=0 then F:=1
else F:=n*F(n-1)
end;
read(m);
```

```

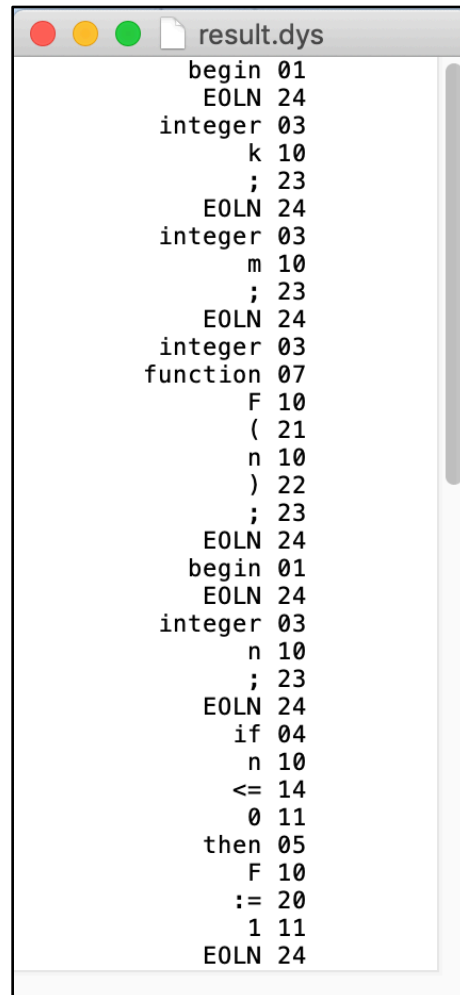
k:=F(m);

write(k)

end

```

本测试程序没有错误产生，二元式序列输出到 result.dys 文件中



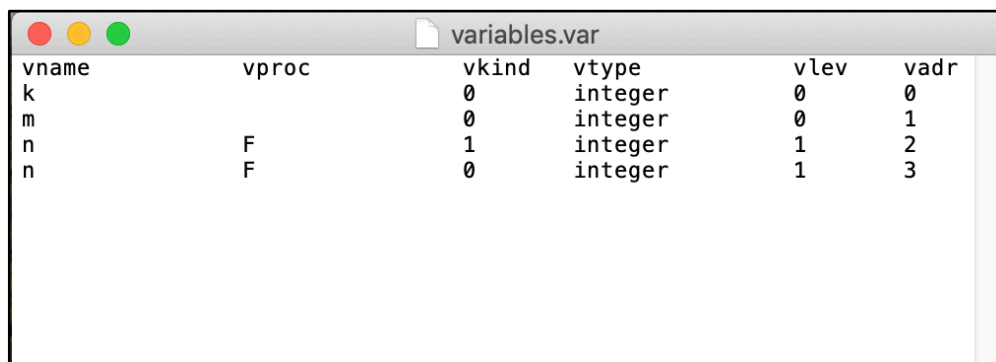
```

begin 01
  EOLN 24
  integer 03
    k 10
    ; 23
  EOLN 24
  integer 03
    m 10
    ; 23
  EOLN 24
  integer 03
function 07
  F 10
  ( 21
    n 10
  ) 22
  ; 23
  EOLN 24
  begin 01
  EOLN 24
  integer 03
    n 10
    ; 23
  EOLN 24
  if 04
    n 10
    <= 14
    0 11
  then 05
    F 10
    := 20
    1 11
  EOLN 24

```

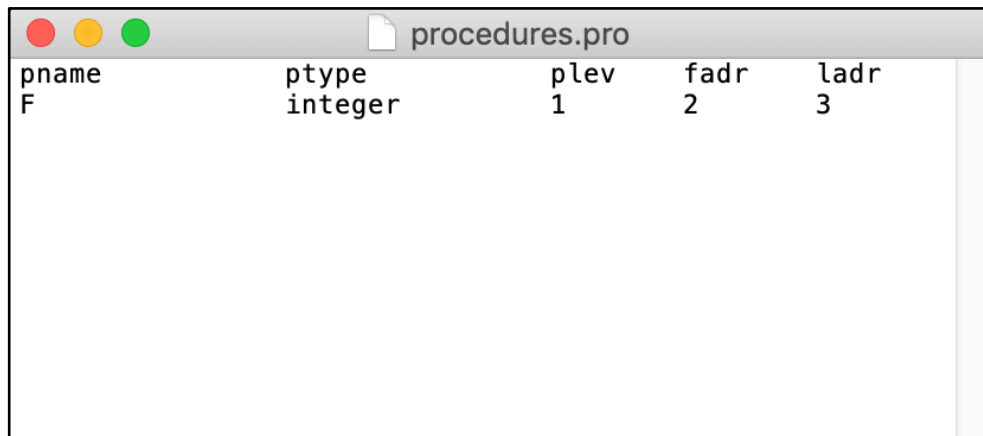
图 3. result.dys 文件节选

相继输出了符号名表 variables.var 和过程名表 procedures.pro



vname	vproc	vkind	vtype	vlev	vadr
k		0	integer	0	0
m		0	integer	0	1
n	F	1	integer	1	2
n	F	0	integer	1	3

图 4. variables.var 符号名表



pname	ptype	plev	fadr	ladr
F	integer	1	2	3

图 5. procedures.pro 过程名表

接着，是具有第一种错误，即缺少符号错的测试程序：

```
begin
  integer k;
  integer m;
  integer function F(n)      // 此处缺少符号: “;”
  begin
    integer n;
    if n<=0 then F:=1
    else F:=n*F(n-1)
  end;
  read(m);
  k:=F(m);
  write(k)
end
```

输出了 error.err 文件



图 6. 缺少符号错的 error.err

接着，是具有第二种错误，即符号匹配错的测试程序：

```
begin
  integer k;
  intege m;           // 此处符号匹配错: intege
  integer function F(n);
  begin
    integer n;
    if n<=0 then F:=1
    else F:=n*F(n-1)
  end;
  read(m);
  k:=F(m);
  write(k)
end
```

输出了 error.err 文件

因为 intege m; 符号匹配错误，按照算法，语法分析器跳到下一行进行分析，所以变量 m 并没有被识别。



图 7. 符号匹配错的 error.err

接着，是具有第三种错误，即符号无定义错的测试程序，即默认测试程序：

```
begin
  integer k;
  integer function F(n);
  begin
```



```

integer n;
if n<=0 then F:=1
else F:=n*F(n-1)
end;
read(m);    // 符号 m 没有定义
k:=F(m);
write(k)
end

```

输出了 error.err 文件



图 7. 符号无定义错的 error.err

接着，是具有第四种错误，即符号重复定义错的测试程序：

```

begin
integer k;
integer m;
integer function F(n);
begin
integer n;
if n<=0 then F:=1
else F:=n*F(n-1)
end;
integer m;    // 符号重复定义
read(m);
k:=F(m);
write(k)

```

end

输出了 error.err 文件



图 8. 符号重复定义错的 error.err

最后，是四种错误都具有的一个测试程序：

```
begin
  integer k;
  integer m;           // 符号匹配错
  integer function F(n) // 缺少符号错
  begin
    integer n;
    if n<=0 then F:=1
    else F:=n*F(n-1)
  end;
  integer k;           // 符号重复定义
  read(m);             // 符号无定义
  k:=F(m);
  write(k)
end
```

最后输出的 error.err 文件是：



图 9. 多种错误下的输出 error.err

## 5. 调试过程

本人在调试环节遇到最大的困难就是对于出错的处理。

对于这三个错误，即缺少符号错，符号无定义错，符号重复定义错，解决起来还算简单，即跳过当前出错符号，根据下一个符号进行下一个符号的处理。

但是对于符号匹配错误，问题就变得棘手，原因是对于符号匹配错误，语法分析器不知道原本想表达的意思是什么。比方说“intege”，词法分析器将它识别为标识符，语法分析器在分析的时候，发现“intege m;”，有两个标识符接连出现，发现语法错误，但是语法分析器并不知道这是一个变量说明文法，而将它识别为了赋值语句，因为只有赋值语句开头是一个标识符。如果跟其它三种错误一样继续分析下一个单词，则之后便会出现一连串错误，因为语法分析器以为在赋值语句的文法中。所以，最后本人就直接跳过此行，进行下一行的识别。这也算是本次实验设计的语法分析器的一个可能改进点。

## 五. 总结心得与建议

本实验进行了对于编译过程中语法分析器的设计。语法分析器的工作就是对于词法分析器输出的二元式序列进行语法分析，检查有没有语法的错误。首先，我们得将原始的二元式序列进行预处理，即单词和种别进行分开，将单词和种别的表输出语法分析器。然后，我们的语法分析器对其进行语法分析，中间生成符号名表和过程名表，如果检测出语法错误，则将错误信息写入出错文件中，如果

没有检测出语法错误，则将二元式序列输出，交给下面的语义分析处理。

本人通过此次实验更好地了解了语法分析器的设计原理以及对于错误的处理方式，感谢老师的讲授。

## 六. 程序说明

/\*

作者:徐贤达

学号:2016060601018

时间:2019/5/8

功能:语法分析器

\*/

/\*

A: 程序

A->B

B: 分程序

B->begin C;M end

C: 说明语句表

C->DC'

C'->; DC'|ε

D: 说明语句

D->E|J

E: 变量说明

E->integer F

F: 变量

F->G

G: 标识符

G->HG'

G'->HG'|IG'|ε

H: 字母

H->a|...|z|A|...|Z

I: 数字

I->0|1|...|9

J: 函数说明

J->integer function G(K);L

K: 参数

K->F

L: 函数体

L->begin C;M end

M: 执行语句表

M->NM'

	$M' \rightarrow ;NM' \epsilon$
N: 执行语句	
	$N \rightarrow O P Q W$
O: 读语句	
	$O \rightarrow \text{read}(F)$
P: 写语句	
	$P \rightarrow \text{write}(F)$
Q: 赋值语句	
	$Q \rightarrow F:=R$
R: 算术表达式	
	$R \rightarrow SR'$
	$R' \rightarrow -SR' \epsilon$
S: 项	
	$S \rightarrow TS'$
	$S' \rightarrow *TS' \epsilon$
T: 因子	
	$T \rightarrow F U Z$
U: 常数	
	$U \rightarrow V$
V: 无符号整数	
	$V \rightarrow IV'$
	$V' \rightarrow IV' \epsilon$
W: 条件语句	
	$W \rightarrow \text{if } X \text{ then } N \text{ else } N$
X: 条件表达式	
	$X \rightarrow RYR$
Y: 关系运算符	
	$Y \rightarrow < <= > >= = <>$
Z: 函数调用	
	$Z \rightarrow G(R)$
*/	

```
import java.util.ArrayList;
import java.io.*;
```

```
public class GrammaticalAnalysis {

    // 存放读入的每个单词
    private static ArrayList<String> input;
    // 存放读入的每个单词对应的种别
    private static ArrayList<String> type;
    // 存放当前单词在 ArrayList 中的 index
    private static int currentToken = 0;
    // 存放变量
```

```

private static ArrayList<Variables> var;
// 指向当前变量
private static Variables currentVar;
// 存放过程
private static ArrayList<Procedures> pro;
// 指向当前过程
private static Procedures currentPro;
// 行号
private static int line = 1;
// 是否产生了语法错误
private static boolean isVirgin = true;
private static BufferedReader bufferedReader;
private static BufferedWriter bufferedWriter1;
private static BufferedWriter bufferedWriter2;
private static BufferedWriter bufferedWriter3;
private static BufferedWriter bufferedWriter4;

private GrammaticalAnalysis() {
    input = new ArrayList<>();
    type = new ArrayList<>();
    var = new ArrayList<>();
    pro = new ArrayList<>();
}

public static void main(String[] args) throws Exception{
    // 初始化语法分析器
    GrammaticalAnalysis grammaticalAnalysis = new GrammaticalAnalysis();
    // 读入文件
    File file = new File("src/Files/result.dyd");
    bufferedReader = new BufferedReader(new FileReader(file));
    String str;
    String temp1;
    String temp2;
    while ((str = bufferedReader.readLine()) != null) {
        temp1= str.substring(0,16);
        temp1 = temp1.trim();
        // 单词存入 input
        input.add(temp1);
        temp2 = str.substring(17);
        // 种别存入 type
        type.add(temp2);
    }
    bufferedReader = new BufferedReader(new FileReader(file));
    File var = new File("src/Files/variables.var");

```

```

bufferedWriter1 = new BufferedWriter(new FileWriter(var));
File pro = new File("src/Files/procedures.pro");
bufferedWriter2 = new BufferedWriter(new FileWriter(pro));
File result = new File("src/Files/result.dys");
bufferedWriter3 = new BufferedWriter(new FileWriter(result));
File error = new File("src/Files/error.err");
bufferedWriter4 = new BufferedWriter(new FileWriter(error));
// 开始执行语法分析
grammaticalAnalysis.A();
// 写变量名表
grammaticalAnalysis.writeVariables();
// 写过程名表
grammaticalAnalysis.writeProcedures();
// 写输出文件
grammaticalAnalysis.writeResults();
bufferedReader.close();
bufferedWriter1.close();
bufferedWriter2.close();
bufferedWriter3.close();
bufferedWriter4.close();
}

/**
 * 输出错误
 * @param errorNumber: 错误种别
 * @param errorInfo: 错误的单词名称
 * @throws Exception
 */
private void printError(int errorNumber, String errorInfo) throws Exception {
    isVirgin = false;
    switch (errorNumber) {
        // 缺少符号错
        case 1:
            System.out.println("ERROR! LINE:" + line + " 缺少符号错:" +
errorInfo + "\n");
            bufferedWriter4.write("ERROR! LINE:" + line + " 缺少符号错:" + "" +
errorInfo + "" + "\n");
            break;
        // 符号匹配错
        case 2:
            System.out.println("ERROR! LINE:" + line + " 符号匹配错:" +
errorInfo + "\n");
            bufferedWriter4.write("ERROR! LINE:" + line + " 符号匹配错:" + "" +
errorInfo + "" + "\n");

```

```

        break;
// 符号无定义
case 3:
    System.out.println("ERROR! LINE:" + line + " 符号无定义:" +
errorInfo + "\n");
    bufferedWriter4.write("ERROR! LINE:" + line + " 符号无定义:" + "" +
errorInfo + "" + "\n");
    break;
// 符号重复定义
case 4:
    System.out.println("ERROR! LINE:" + line + " 符号重复定义:" +
errorInfo + "\n");
    bufferedWriter4.write("ERROR! LINE:" + line + " 符号重复定义:" + ""
+ errorInfo + "" + "\n");
    break;
default:
    break;
    }
}

/**
 * 获取下一个单词
 */
private void nextToken() {
    currentToken++;
    while (input.get(currentToken).equals("EOLN")) {
        currentToken++;
        line++;
    }
}

/**
 * 获取下一个单词在 input 中的 index
 * @return 下一个单词在 input 中的 index
 */
private int getNextToken() {
    int nextToken = currentToken + 1;
    while (input.get(nextToken).equals("EOLN"))
        nextToken++;
    return nextToken;
}

/**
 * 跳到下一行(符号匹配错下)

```



```

    */
private void jumpToNextLine() {
    while (!input.get(currentToken).equals("EOLN"))
        currentToken++;
    currentToken++;
}

/**
 * 判断变量名是否合法
 * @param vname:变量名
 * @param vproc:变量所属过程名称
 * @param vkind:变量种别
 * @return 是与否
 */
private boolean isVarIllegal(String vname, String vproc, int vkind) {
    if (var.size() == 0 || pro.size() == 0)
        return false;
    for (int i = 0; i < pro.size(); i++) {
        if (vname.equals(pro.get(i).pname))
            return true;
    }
    for (int i = 0; i < var.size(); i++) {
        if (vname.equals(var.get(i).vname) && vproc.equals(var.get(i).vproc) &&
vkind == var.get(i).vkind)
            return true;
    }
    return false;
}

/**
 * 判断过程名是否合法
 * @param pname:过程名
 * @return 是与否
 */
private boolean isProIllegal(String pname) {
    if (var.size() == 0 || pro.size() == 0)
        return false;
    for (int i = 0; i < pro.size(); i++) {
        if (pname.equals(pro.get(i).pname))
            return true;
    }
    for (int i = 0; i < var.size(); i++) {
        if (pname.equals(var.get(i).vname))
            return true;
    }
}

```

```

    }
    return false;
}

/**
 * 写变量名表
 * 左对齐
 * @throws Exception
 */
private void writeVariables() throws Exception {
    bufferedWriter1.write(String.format("%-16s", "vname"));
    bufferedWriter1.write(String.format("%-16s", "vproc"));
    bufferedWriter1.write(String.format("%-8s", "vkind"));
    bufferedWriter1.write(String.format("%-16s", "vtype"));
    bufferedWriter1.write(String.format("%-8s", "vlev"));
    bufferedWriter1.write(String.format("%-8s", "vadr"));
    bufferedWriter1.write("\n");
    System.out.println("Variables");
    System.out.print(String.format("%-16s", "vname"));
    System.out.print(String.format("%-16s", "vproc"));
    System.out.print(String.format("%-8s", "vkind"));
    System.out.print(String.format("%-16s", "vtype"));
    System.out.print(String.format("%-8s", "vlev"));
    System.out.print(String.format("%-8s", "vadr"));
    System.out.print("\n");
    for (int i = 0; i < var.size(); i++) {
        bufferedWriter1.write(String.format("%-16s", var.get(i).vname));
        bufferedWriter1.write(String.format("%-16s", var.get(i).vproc));
        bufferedWriter1.write(String.format("%-8s", var.get(i).vkind));
        bufferedWriter1.write(String.format("%-16s", var.get(i).vtype.toString()));
        bufferedWriter1.write(String.format("%-8s", var.get(i).vlev));
        bufferedWriter1.write(String.format("%-8s", var.get(i).vadr));
        bufferedWriter1.write("\n");
        System.out.print(String.format("%-16s", var.get(i).vname));
        System.out.print(String.format("%-16s", var.get(i).vproc));
        System.out.print(String.format("%-8s", var.get(i).vkind));
        System.out.print(String.format("%-16s", var.get(i).vtype.toString()));
        System.out.print(String.format("%-8s", var.get(i).vlev));
        System.out.print(String.format("%-8s", var.get(i).vadr));
        System.out.print("\n");
    }
    System.out.println();
}

```

```

/**
 * 写过程名表
 * 左对齐
 * @throws Exception
 */
private void writeProcedures() throws Exception {
    bufferedWriter2.write(String.format("%-16s", "pname"));
    bufferedWriter2.write(String.format("%-16s", "ptype"));
    bufferedWriter2.write(String.format("%-8s", "plev"));
    bufferedWriter2.write(String.format("%-8s", "fadr"));
    bufferedWriter2.write(String.format("%-8s", "ladr"));
    bufferedWriter2.write("\n");
    System.out.println("Procedures");
    System.out.print(String.format("%-16s", "pname"));
    System.out.print(String.format("%-16s", "ptype"));
    System.out.print(String.format("%-8s", "plev"));
    System.out.print(String.format("%-8s", "fadr"));
    System.out.print(String.format("%-8s", "ladr"));
    System.out.print("\n");
    for (int i = 0; i < pro.size(); i++) {
        bufferedWriter2.write(String.format("%-16s", pro.get(i).pname));
        bufferedWriter2.write(String.format("%-16s", pro.get(i).ptype.toString()));
        bufferedWriter2.write(String.format("%-8s", pro.get(i).plev));
        bufferedWriter2.write(String.format("%-8s", pro.get(i).fadr));
        bufferedWriter2.write(String.format("%-8s", pro.get(i).ladr));
        bufferedWriter2.write("\n");
        System.out.print(String.format("%-16s", pro.get(i).pname));
        System.out.print(String.format("%-16s", pro.get(i).ptype.toString()));
        System.out.print(String.format("%-8s", pro.get(i).plev));
        System.out.print(String.format("%-8s", pro.get(i).fadr));
        System.out.print(String.format("%-8s", pro.get(i).ladr));
        System.out.print("\n");
    }
    System.out.println();
}

/**
 * 写输出结果
 * @throws Exception
 */
private void writeResults() throws Exception {
    if (isVirgin) {
        String str;
        while ((str = bufferedReader.readLine()) != null) {

```

```

        bufferedWriter3.write(str + "\n");
    }
} else {
    bufferedWriter3.write("Unable to pass the grammatical analysis!");
}
}

/**
 * A:程序
 * A->B
 * @throws Exception
 */
private void A() throws Exception {
    currentPro = new Procedures();
    B();
}

/**
 * B:分程序
 * B->begin C;M end
 * @throws Exception
 */
private void B() throws Exception {
    if (input.get(currentToken).equals("begin")) {
        nextToken();
    } else {
        // 出错处理:跳到 C
        printError(1, "begin");
        if (!input.get(currentToken).equals("integer"))
            nextToken();
    }
    C();
    if (input.get(currentToken).equals(";")) {
        nextToken();
    } else {
        // 出错处理:跳到 M
        printError(1, ";");
        if (((!input.get(currentToken).equals("integer")) &&
            (!input.get(currentToken).equals("read")) &&
            (!input.get(currentToken).equals("write")) &&
            (!type.get(currentToken).equals("10")))) {
            nextToken();
        }
    }
}

```

```

        M();
        if (input.get(currentToken).equals("end")) {
            nextToken();
        } else {
            printError(1, "end");
        }
    }
}

/**
 * C:说明语句表(左递归)
 * C->DC'
 * @throws Exception
 */
private void C() throws Exception {
    D();
    C_();
}

/**
 * C'
 * C'->;DC'|ε
 * @throws Exception
 */
private void C_() throws Exception {
    int nextToken = getNextToken();
    if (input.get(currentToken).equals(";") && input.get(nextToken).equals("integer")) {
        nextToken();
        D();
        C_();
    }
    else if (input.get(currentToken).equals(";") && type.get(nextToken).equals("10")) {
        nextToken();
        // 出错处理:跳过一行
        printError(2, input.get(currentToken));
        jumpToNextLine();
        D();
        C_();
    }
    else {
        if (input.get(currentToken).equals("integer")) {
            // 出错处理:跳到 D
            printError(1, ";");
            D();
            C_();
        }
    }
}

```

```

    }
}

/**
 * D:说明语句
 * D->E|J
 * @throws Exception
 */
private void D() throws Exception {
    if (input.get(getNextToken()).equals("function"))
        J();
    else
        E();
}

/**
 * E:变量说明
 * E->integer F
 * @throws Exception
 */
private void E() throws Exception {
    if (input.get(currentToken).equals("integer"))
        nextToken();
    else {
        printError(1, "integer");
        nextToken();
    }
    // 创建新变量(实参)
    currentVar = new Variables();
    currentVar.vname = input.get(currentToken);
    currentVar.vproc = currentPro.pname;
    currentVar.vkind = 0;
    currentVar.vtype = Types.integer;
    currentVar.vlev = currentPro.plev;
    currentVar.vadr = var.size();
    if (isVarIllegal(currentVar.vname, currentVar.vproc, currentVar.vkind))
        printError(4, currentVar.vname);
    else {
        if (currentPro.varNum == 0)
            currentPro.fadr = currentVar.vadr;
        currentPro.ladr = currentVar.vadr;
        currentPro.varNum++;
        var.add(currentVar);
    }
}

```

```

        F();
    }

    /**
     * F:变量
     * F->G
     * @throws Exception
     */
    private void F() throws Exception {
        G();
    }

    /**
     * G:标识符(左递归)
     * G->HG'
     * @throws Exception
     */
    private void G() throws Exception {
        if (type.getCurrentToken().equals("10"))
            nextToken();
    }

    /**
     * G'
     * G'->HG'|IG'|ε
     * @throws Exception
     */
    private void G_() throws Exception {}

    /**
     * H:字母
     * H->a|...|z|A|...|Z
     * @throws Exception
     */
    private void H() throws Exception {}

    /**
     * I:数字
     * I->0|1|...|9
     * @throws Exception
     */
    private void I() throws Exception {}

    /**

```

```

* J:函数说明
* J->integer function G(K);L
* @throws Exception
*/
private void J() throws Exception {
    // 创建新进程
    Procedures temp = new Procedures();
    temp.pname = currentPro.pname;
    temp.ptype = currentPro.ptype;
    temp.plev = currentPro.plev;
    temp.fadr = currentPro.fadr;
    temp.ladr = currentPro.ladr;
    temp.varNum = currentPro.varNum;
    if (input.get(currentToken).equals("integer"))
        nextToken();
    else {
        // 出错处理:跳到:"function"
        printError(1, "integer");
        if (!input.get(currentToken).equals("function"))
            nextToken();
    }
    if (input.get(currentToken).equals("function"))
        nextToken();
    else {
        // 出错处理:跳到 G
        printError(1, "function");
        if (!type.get(currentToken).equals("10"))
            nextToken();
    }
    currentPro.pname = input.get(currentToken);
    currentPro.ptype = Types.integer;
    currentPro.plev++;
    currentPro.varNum = 0;
    if (isProIllegal(input.get(currentToken)))
        printError(4, currentPro.pname);
    G();
    if (input.get(currentToken).equals("("))
        nextToken();
    else {
        // 出错处理:跳到 K
        printError(1, "(");
        if (!type.get(currentToken).equals("10"))
            nextToken();
    }
}

```



```

        K();
        if (input.get(currentToken).equals(""))
            nextToken();
        else {
            // 出错处理:跳到";"
            printError(1, "");
            if (!input.get(currentToken).equals(";"))
                nextToken();
        }
        if (input.get(currentToken).equals(";"))
            nextToken();
        else {
            // 出错处理:跳到 L
            printError(1, "");
            if (!input.get(currentToken).equals("begin"))
                nextToken();
        }
        L();
        currentPro = temp;
    }

/**
 * K:参数
 * K->F
 * @throws Exception
 */
private void K() throws Exception {
    // 创建新变量(形参)
    currentVar = new Variables();
    currentVar.vname = input.get(currentToken);
    currentVar.vproc = currentPro.pname;
    currentVar.vkind = 1;
    currentVar.vtype = Types.integer;
    currentVar.vlev = currentPro.plev;
    currentVar.vadr = var.size();
    if (isVarIllegal(currentVar.vname, currentVar.vproc, currentVar.vkind))
        printError(4, currentVar.vname);
    else {
        if (currentPro.varNum == 0)
            currentPro.fadr = currentVar.vadr;
        currentPro.ladr = currentVar.vadr;
        currentPro.varNum++;
        var.add(currentVar);
    }
}

```

```

        F();
    }

/**
 * L:函数体
 * L->begin C;M end
 * @throws Exception
 */
private void L() throws Exception {
    if (input.getCurrentToken().equals("begin"))
        nextToken();
    else {
        // 出错处理:跳到 C
        printError(1, "begin");
        if (!input.getCurrentToken().equals("integer"))
            nextToken();
    }
    C();
    pro.add(currentPro);
    if (input.getCurrentToken().equals(";"))
        nextToken();
    else {
        // 出错处理:跳到 M
        printError(1, ";");
        if ((!input.getCurrentToken().equals("integer")) &&
            (!input.getCurrentToken().equals("read")) &&
            (!input.getCurrentToken().equals("write")) &&
            (!type.getCurrentToken().equals("10"))) {
            nextToken();
        }
    }
    M();
    if (input.getCurrentToken().equals("end"))
        nextToken();
    else {
        // 出错处理:跳到"end"or";"
        printError(1, "end");
        if ((!input.getCurrentToken().equals(";")) &&
            (!input.getCurrentToken().equals("end")))
            nextToken();
    }
}

/**

```

```

* M:执行语句表(左递归)
* M->NM'
* @throws Exception
*/
private void M() throws Exception {
    N();
    M_();
}

/**
* M'
* M'->;NM'|ε
* @throws Exception
*/
private void M_() throws Exception {
    if (input.getCurrentToken().equals(";")) {
        nextToken();
        N();
        M_();
    } else {
        if (!input.getCurrentToken().equals("end") &&
            (!input.getCurrentToken().equals("EOF"))) {
            // 出错处理:跳到"end"
            printError(1, ";");
            N();
            M_();
        }
    }
}

/**
* N:执行语句
* N->O|P|Q|W
* @throws Exception
*/
private void N() throws Exception {
    if (input.getCurrentToken().equals("read"))
        O();
    else if (input.getCurrentToken().equals("write"))
        P();
    else if (input.getCurrentToken().equals("if"))
        W();
    else if (type.getCurrentToken().equals("10"))
        Q();
}

```

```

        else {
            printError(2, input.get(currentToken));
            nextToken();
        }
    }

/**
 * O:读语句
 * O->read(F)
 * @throws Exception
 */
private void O() throws Exception {
    if (input.get(currentToken).equals("read"))
        nextToken();
    else {
        // 出错处理:跳到 "("
        printError(1, "read");
        if (!input.get(currentToken).equals("("))
            nextToken();
    }
    if (input.get(currentToken).equals("("))
        nextToken();
    else {
        // 出错处理:跳到 F
        printError(1, "(");
        if (!type.get(currentToken).equals("10"))
            nextToken();
    }
    if (!isVarIllegal(input.get(currentToken), currentPro.pname, 0) &&
        !isVarIllegal(input.get(currentToken), currentPro.pname, 1))
        printError(3, input.get(currentToken));
    F();
    if (input.get(currentToken).equals(""))
        nextToken();
    else {
        // 出错处理:跳到 ";"或"end"
        printError(1, "");
        if ((!input.get(currentToken).equals(";")) &&
            !input.get(currentToken).equals("end"))
            nextToken();
    }
}

/**

```

```

* P:写语句
* P->write(F)
* @throws Exception
*/
private void P() throws Exception {
    if (input.get(currentToken).equals("write"))
        nextToken();
    else {
        // 出错处理:跳到 "("
        printError(1, "write");
        if (!input.get(currentToken).equals("("))
            nextToken();
    }
    if (input.get(currentToken).equals("("))
        nextToken();
    else {
        // 出错处理:跳到 F
        printError(1, "(");
        if (!input.get(currentToken).equals("10"))
            nextToken();
    }
    if ((!isVarIllegal(input.get(currentToken), currentPro.pname, 0)) &&
        (!isVarIllegal(input.get(currentToken), currentPro.pname, 1)))
        printError(3, input.get(currentToken));
    F();
    if (input.get(currentToken).equals(""))
        nextToken();
    else {
        // 出错处理:跳到 "end"
        printError(1, "");
        if ((!input.get(currentToken).equals(";")) &&
            (!input.get(currentToken).equals("end")))
            nextToken();
    }
}

/**
* Q:赋值语句
* Q->F:=R
* @throws Exception
*/
private void Q() throws Exception {
    if ((!isVarIllegal(input.get(currentToken), currentPro.pname, 0)) &&
        (!isVarIllegal(input.get(currentToken), currentPro.pname, 1))) {

```

```

        printError(3, input.get(currentToken));
    }
    F();
    if (input.get(currentToken).equals(":="))
        nextToken();
    else {
        // 出错处理:跳到 R
        printError(1, ":=");
        if ((!type.get(currentToken).equals("10")) &&
            (!type.get(currentToken).equals("11")))
            nextToken();
    }
    R();
}

/**
 * R:算术表达式(左递归)
 * R->SR'
 * @throws Exception
 */
private void R() throws Exception {
    S();
    R_();
}

/**
 * R'
 * R'->-SR'|ε
 * @throws Exception
 */
private void R_() throws Exception {
    if (input.get(currentToken).equals("-")) {
        nextToken();
        S();
        R_();
    } else {
        if (type.get(currentToken).equals("10") || type.get(currentToken).equals("11"))
        {
            S();
            R_();
        }
    }
}

```

```

/**
 * S:项(左递归)
 * S->TS'
 * @throws Exception
 */
private void S() throws Exception {
    T();
    S_();
}

/**
 * S'
 * S'->*TS'|ε
 * @throws Exception
 */
private void S_() throws Exception {
    if (input.getCurrentToken().equals("*")) {
        nextToken();
        T();
        S_();
    } else {
        if (type.getCurrentToken().equals("10") || type.getCurrentToken().equals("11"))
        {
            T();
            S_();
        }
    }
}

/**
 * T:因子
 * T->F|U|Z
 * @throws Exception
 */
private void T() throws Exception {
    if (type.getCurrentToken().equals("11"))
        U();
    else if (input.getNextToken().equals("("))
        Z();
    else {
        if ((!isVarIllegal(input.getCurrentToken(), currentPro.pname, 0)) &&
            (!isVarIllegal(input.getCurrentToken(), currentPro.pname, 1)))
            printError(3, input.getCurrentToken());
        F();
    }
}

```

```

    }
}

/**
 * U:常数
 * U->V
 * @throws Exception
 */
private void U() throws Exception {
    if (type.getCurrentToken().equals("11"))
        nextToken();
}

/**
 * V:无符号整数(左递归)
 * V->IV'
 * @throws Exception
 */
private void V() throws Exception {}

/**
 * V'
 * V'->IV'|ε
 * @throws Exception
 */
private void V_() throws Exception {}

private void W() throws Exception {
    if (input.getCurrentToken().equals("if"))
        nextToken();
    else {
        // 出错处理:跳到 X
        printError(1, "if");
        if ((!type.getCurrentToken().equals("10")) &&
            !(type.getCurrentToken().equals("11")))
            nextToken();
    }
    X();
    if (input.getCurrentToken().equals("then"))
        nextToken();
    else {
        printError(1, "then"); // N
        if ((!input.getCurrentToken().equals("integer")) &&
            (!input.getCurrentToken().equals("read")) &&

```



```

                (!input.get(currentToken).equals("write")) &&
                (!type.get(currentToken).equals("10"))) {
            nextToken();
        }
    }
    N();
    if (input.get(currentToken).equals("else"))
        nextToken();
    else {
        printError(1, "else"); // N
        if ((!input.get(currentToken).equals("integer")) &&
            (!input.get(currentToken).equals("read")) &&
            (!input.get(currentToken).equals("write")) &&
            (!type.get(currentToken).equals("10"))) {
            nextToken();
        }
    }
    N();
}

```

```

/**
 * X:条件表达式
 * X->R Y R
 * @throws Exception
 */
private void X() throws Exception {
    R();
    Y();
    R();
}

```

```

/**
 * Y:关系运算符
 * Y-><|<=|>|>=|=|<>
 * @throws Exception
 */
private void Y() throws Exception {
    if (input.get(currentToken).equals("<") ||
        input.get(currentToken).equals("<=") ||
        input.get(currentToken).equals(">") ||
        input.get(currentToken).equals(">=") ||
        input.get(currentToken).equals("=>") ||
        input.get(currentToken).equals("=<"))
        nextToken();
}

```

```

        else {
            // 出错处理:跳到 R
            printError(1, "运算符");
            if ((!type.getCurrentToken().equals("10")) &&
                (!type.getCurrentToken().equals("11")))
                nextToken();
        }
    }

/**
 * Z:函数调用
 * Z->G(R)
 * @throws Exception
 */
private void Z() throws Exception {
    if (!isProIllegal(input.getCurrentToken()))
        printError(3, input.getCurrentToken());
    G();
    if (input.getCurrentToken().equals("("))
        nextToken();
    else {
        // 出错处理:跳到 R
        printError(1, "(");
        if ((!type.getCurrentToken().equals("10")) &&
            (!type.getCurrentToken().equals("11")))
            nextToken();
    }
    R();
    if (input.getCurrentToken().equals("("))
        nextToken();
    else {
        // 出错处理:跳到 R
        printError(1, "(");
        if ((!input.getCurrentToken().equals("-")) &&
            (!input.getCurrentToken().equals("*")) &&
            (!input.getCurrentToken().equals(";")) &&
            (!input.getCurrentToken().equals("end")))
            nextToken();
    }
}
}
}

```