编号：

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实习 | 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 九 | 十 | 总评 | 教师签名 |
| 成绩 |  |  |  |  |  |  |  |  |  |  |  |  |

武汉大学计算机学院

《编译原理》课程

语法分析

实习报告

编 号：

实习题目： 构造小语言的语法分析程序

专业（班）： 2018级计科7班

学生学号： 2018302100026

学生姓名： 龙 晓 怡

任课教师： 杜　卓　敏

2021年6月

目 录

第一部分 语言语法规则 ……………………… 1

第二部分 文法定义 ………………………

第三部分 语法分析算法 ………………………

第四部分 出错处理出口 ………………………

第五部分 测试计划 ………………………

目录

[关键变量定义 V](#_Toc73195118)

实验目的：构造一个Mini语言的语法分析程序

实验要求：

要求Mini语言至少包含算算术表达式、逻辑表达式、赋值语句、if-then分支结构、while循环结构等基本控制结构。

输入：Mini语言源程序

输出：与该源程序是否符合语法规则要求的错误报告

定义Mini语言语法成分

定义Mini语言语法规则（非左递归文法描述语法规则

设计各语法成分分析子程序（递归下降分析方法

出错处理的出口

设计典型测试用例

词法分析是编译程序进行编译时第一个要进行的任务，主要是对源程序进行编译预处理（去除注释、无用的回车换行找到包含的文件等）之后，对整个源程序进行分解，分解成一个个单词，这些单词有且只有五类，分别是标识符、保留字、常数、运算符、界符。以便为下面的语法分析和语义分析做准备。可以说词法分析面向的对象是单个的字符，目的是把它们组成有效的单词（字符串）

而语法的分析则是依据语法规则，逐一分析词法分析时得到的单词串，把单词串分解成各类语法单位，即确定它们是怎样组成说明和语句，以及说明和语句又是怎样组成程序的。分析时如发现有不合语法规则的地方，便将出错的位置及出错性质打印报告给程序员；如无语法错误，则用另一种中间形式给出正确的语法结构，供下一阶段分析使用。

递归下降分析方法（递归子程序法）

1 基本思想：

对每一个语法成分（用非终结符号代表），构造相应的分析子程序，该分析子程序分析相应于该语法成分（非终结符号）的符号串。

由于语法成分之间不可避免会含有递归，所以分析子程序之间也会有递归调用，故而又称为递归子程序法

2 分析过程

从开始符号出发，在语法规则支配下，逐个扫描输入符号串中的符号，根据文法和当前的输入符号预测到下一个语法成分是U时，便确定U为目标，并调用U的分析子程序P(U)工作。在P(U)工作的过程中，又有可能确定U或其它非终结符号为子目标，并调用相应的分析子程序。如此继续下去，直到得到结果。

词法分析阶段

输入源程序

输出二元式序列文件\*.dyd作为语法分析器的输入

语法分析阶段

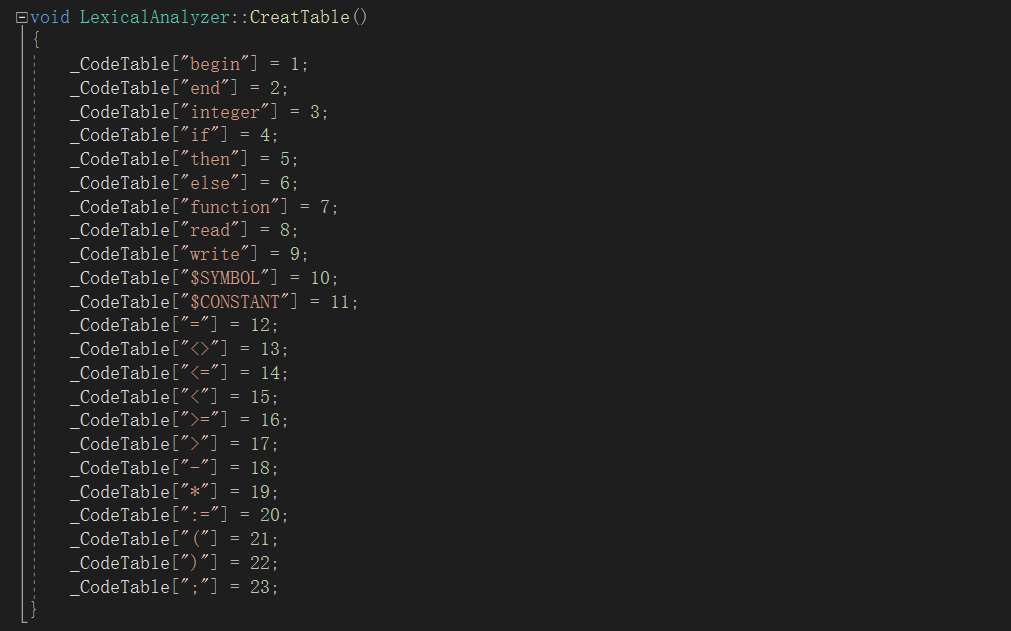
输入词法分析结果

输出源程序错误报告error.err

# 主要流程分析

## 定义词法规则

### Mini语言词汇表（类别和表现形式）



### 根据词法规则写出正规文法

<程序>→<分程序>

<分程序>→begin <说明语句表>；<执行语句表> end

<说明语句表>→<说明语句>│<说明语句表> ；<说明语句>

<说明语句>→<变量说明>│<函数说明>

<变量说明>→integer <变量>

<变量>→<标识符>

<标识符>→<字母>│<标识符><字母>│ <标识符><数字>

<字母>→a│b│c│d│e│f│g│h│i│j│k│l│m│n│o │p│q │r│s│t│u│v│w│x│y│z

<数字>→0│1│2│3│4│5│6│7│8│9

<函数说明>→integer function <标识符>（<参数>）；<函数体>

<参数>→<变量>

<函数体>→begin <说明语句表>；<执行语句表> end

<执行语句表>→<执行语句>│<执行语句表>；<执行语句>

<执行语句>→<读语句>│<写语句>│<赋值语句>│<条件语句>

<读语句>→read(<变量>)

<写语句>→write(<变量>)

<赋值语句>→<变量>:=<算术表达式>

<算术表达式>→<算术表达式>-<项>│<项>

<项>→<项>\*<因子>│<因子>

<因子>→<变量>│<常数>│<函数调用>

<常数>→<无符号整数>

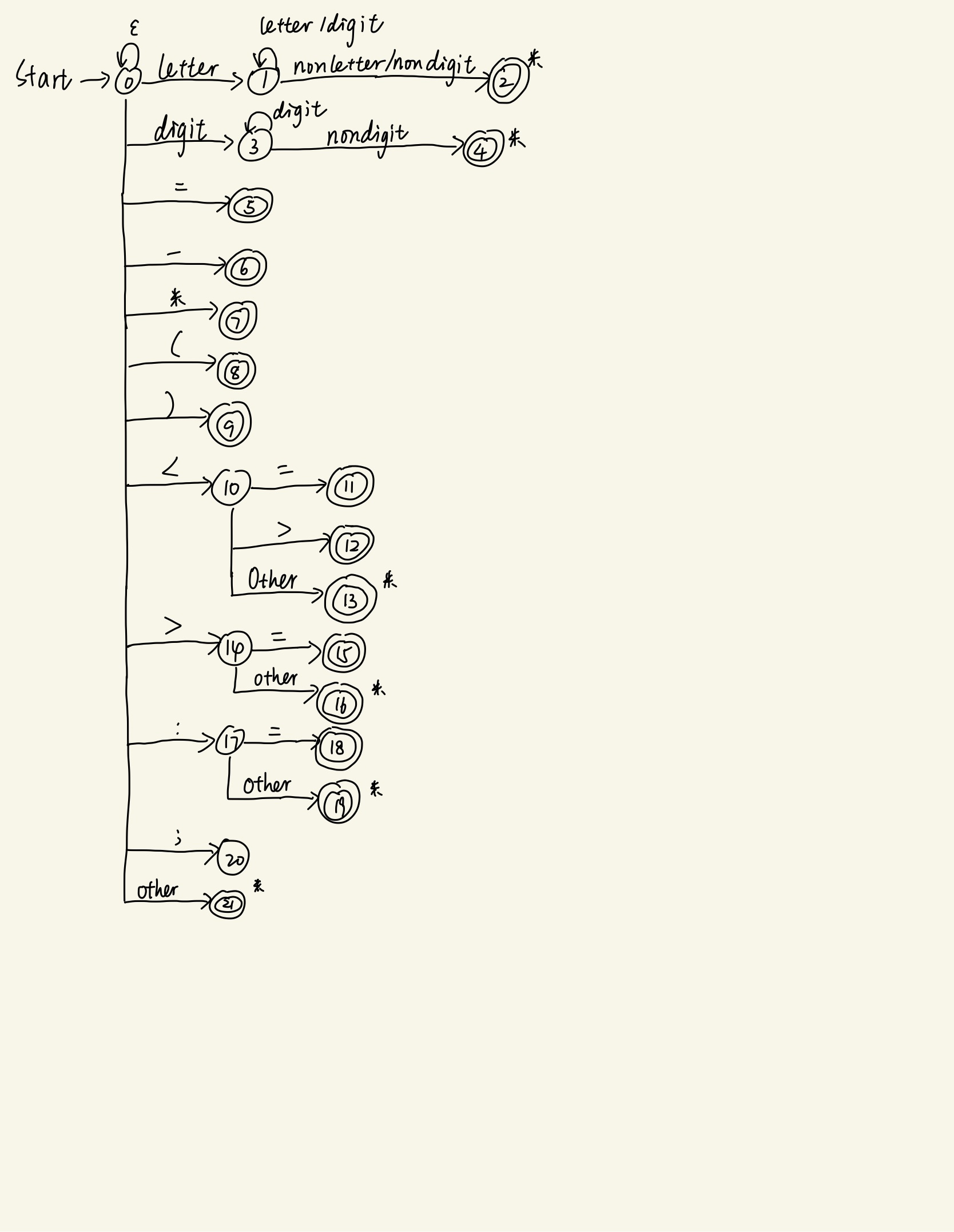
<无符号整数>→<数字>│<无符号整数><数字>

<条件语句>→if<条件表达式>then<执行语句>else <执行语句>

<条件表达式>→<算术表达式><关系运算符><算术表达式>

<关系运算符> →<│<=│>│>=│=│<>

### 将正规文法转换为状态图

将初始状态合并为唯一的初态

### 改进文法（非左递归描述

改进文法G'如下：

S→begin A B end

A→C;A'

A'→C;A'|ε

C→integer D

D→<标识符> | function <标识符>（<标识符>）；E

E→begin A B end

B→FB'

B'→;FB'|ε

F→read(<标识符>)│write(<标识符>)│G│H

G→<标识符>:=I

I→JI'

I'→-JI'|ε

J→KJ'

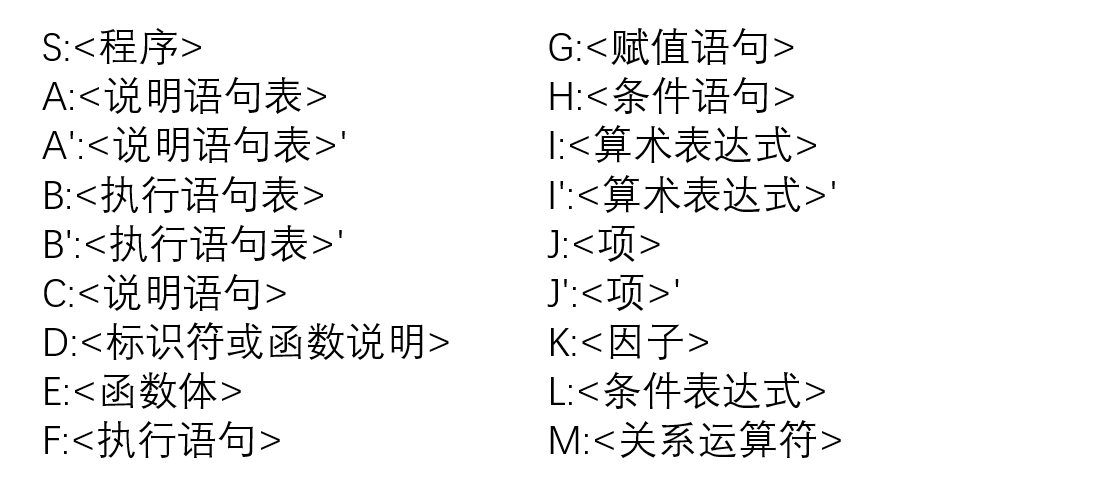
J'→\*KJ' | ε

K→<标识符>│<常数>│<函数调用>

H→if L then F else F

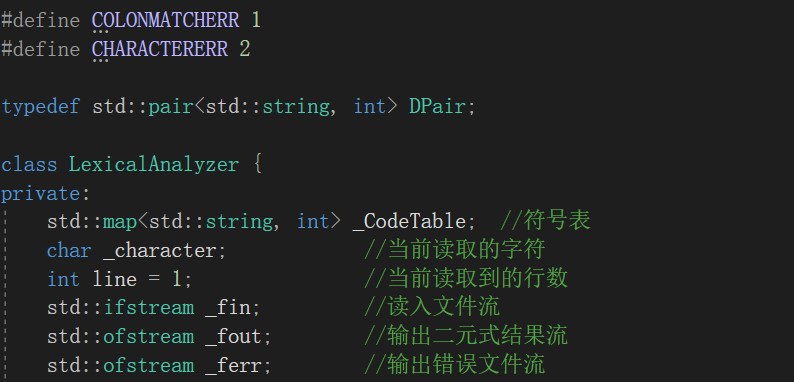
L→IMI

M→<│<=│>│>=│=│<>

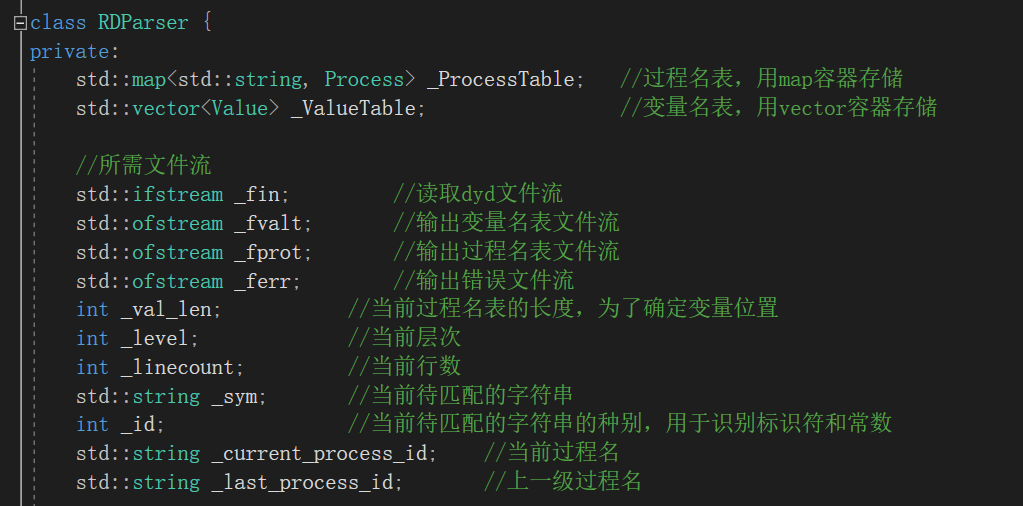
其中：

# 关键变量定义

## 词法分析关键变量



## 语法分析关键变量

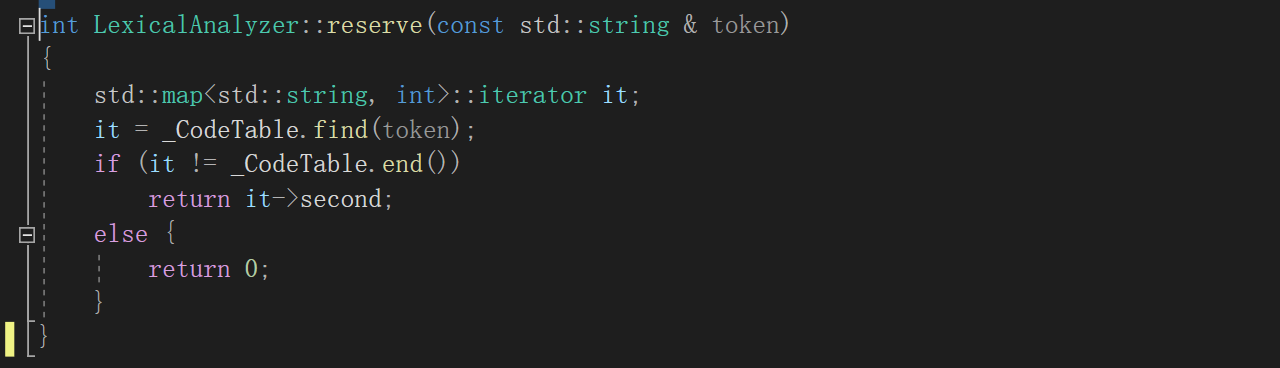


# 关键函数定义

## 词法分析部分关键函数

### 查找保留字int reserve(const std::string &)

若查找成功，返回种别码

若查找不成功，返回0，即为标识符

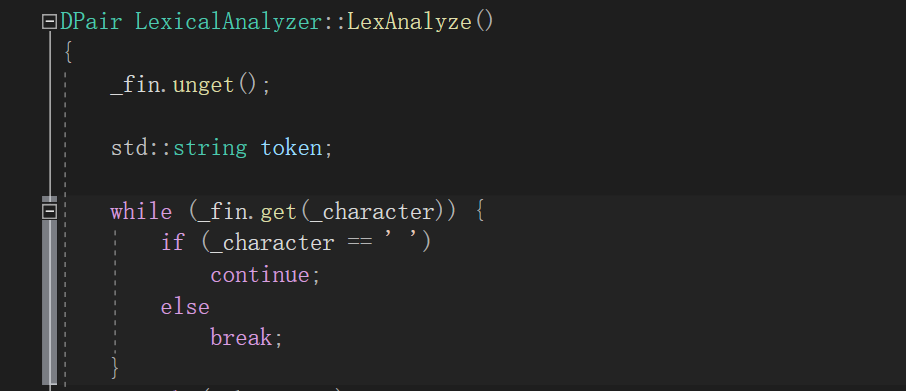
### 词法分析器DPair LexAnalyze()

该词法分析器完成的主要功能

1）跳过空白。即对于空格，\t，\n字符，需要自动跳过；

2）判断读入的字符（或字符串）是否为某类词法单元；

3）通过若干判断分支，最终得到的返回值即为识别出的一个词法单元.

预处理：跳过空白

分析标识符

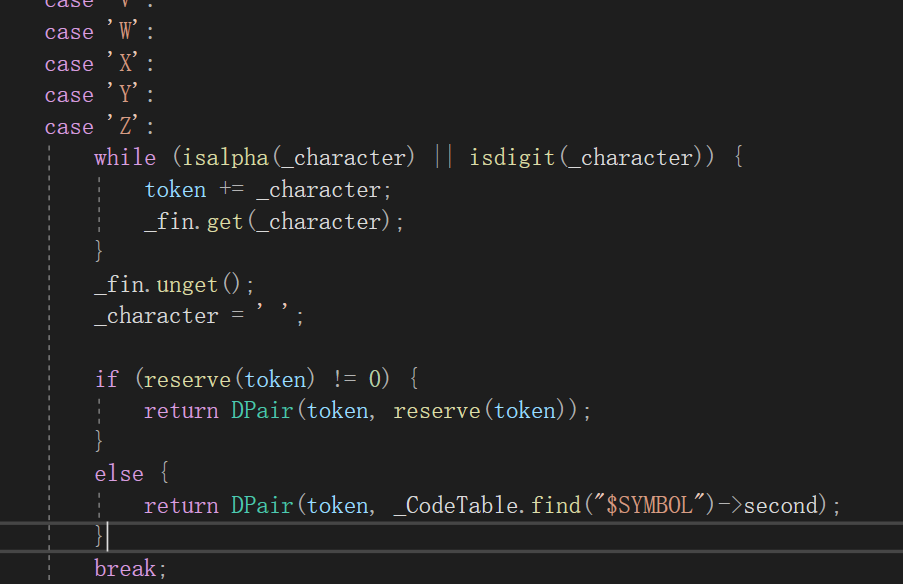
程序到达这里的状态是：当前字符一定是英文字符，且当前字符的前一个字符一定不是英文字符

标识符包含关键字和变量名

注意：

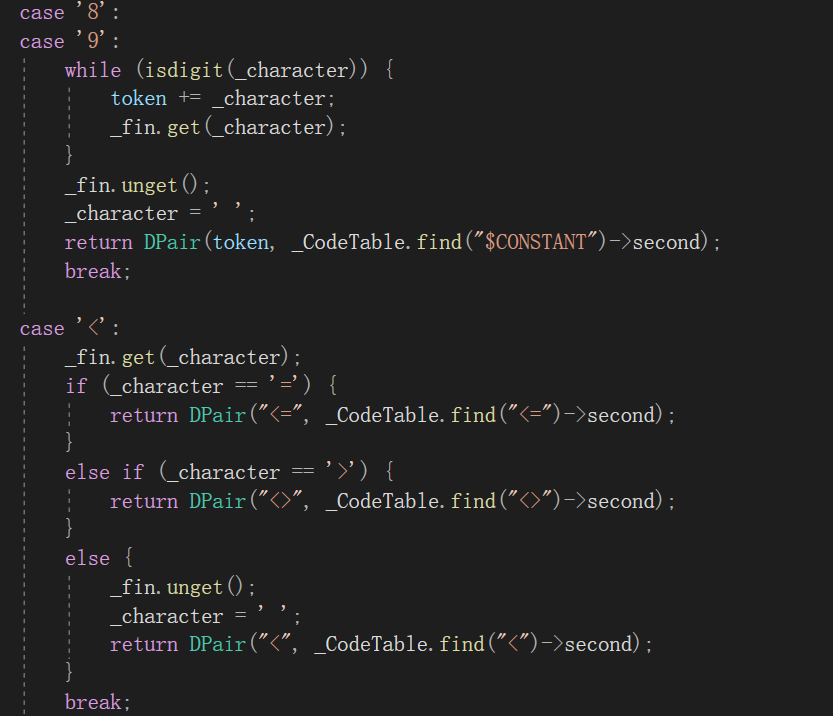
1）对于变量名，开头的一定是字母，接下来可以是字母或数字

2）如果识别出一个标识符全部由字母组成，且不属于任何关键字，则归为变量名，否则归为关键字



分析整型常量

程序到达这里状态是：当前字符一定是数字，且当前字符的前一个字符一定不是数字。

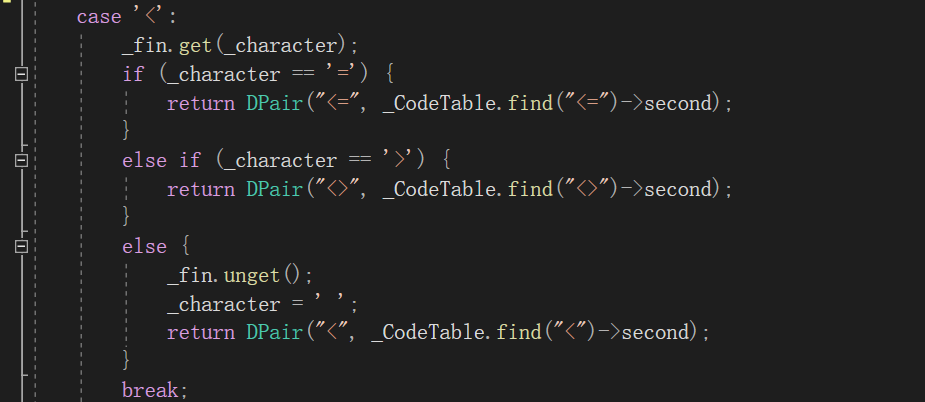
如果要识别完整的整数，需要不断读入字符并保存起来，直到当前字符不是数字为止。

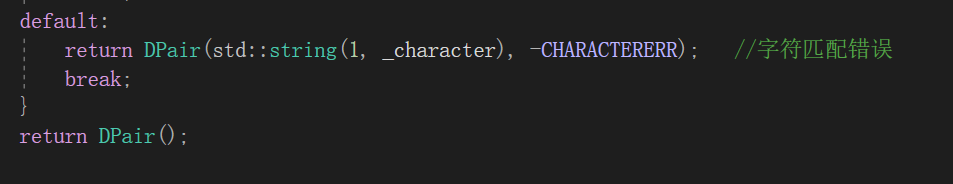
分析操作符

程序到达这里的状态是：当前字符一定是合法操作符的一部分，且当前字符的前一个字符一定不属于任何合法的操作符。

注意：

1)识别操作符需要最长串匹配原则，如下例读到字符‘<’时，再读入一个字符即可判断是否构成操作符‘<=’, ‘<>’, 如果不匹配，把字符rollback，利用unget()函数实现。



均不匹配，遇到非法字符报错

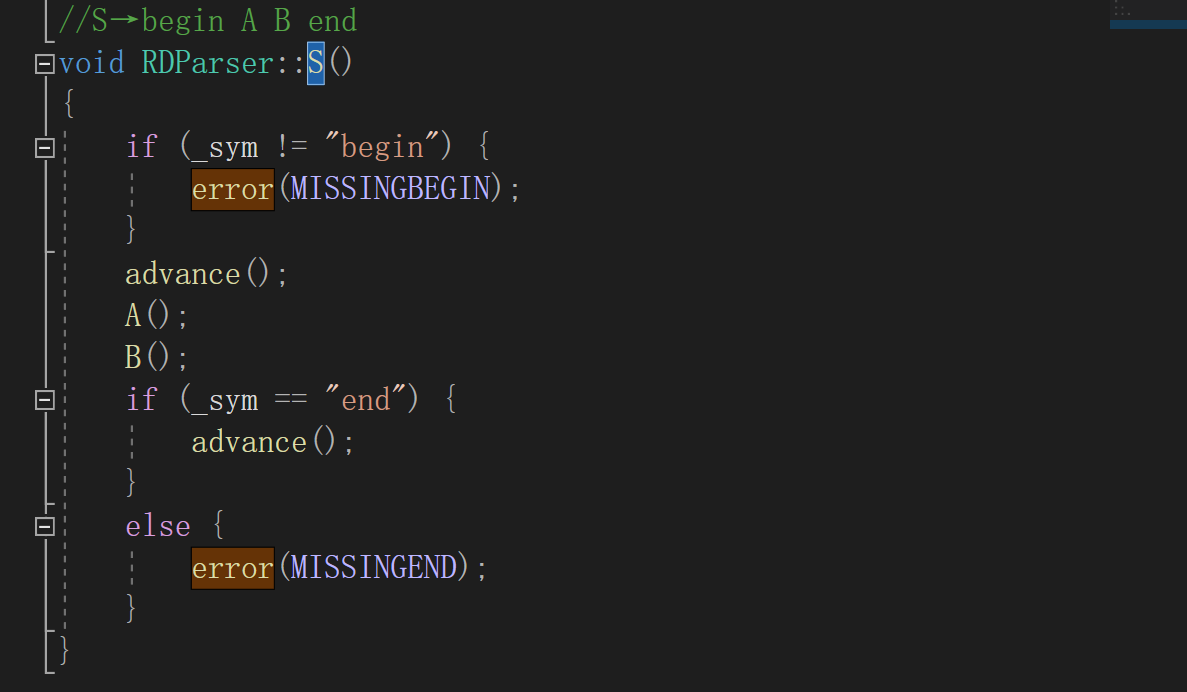
### 报错函数void error(int type)

### 输出词法分析结果函数void output(const DPair binseq)

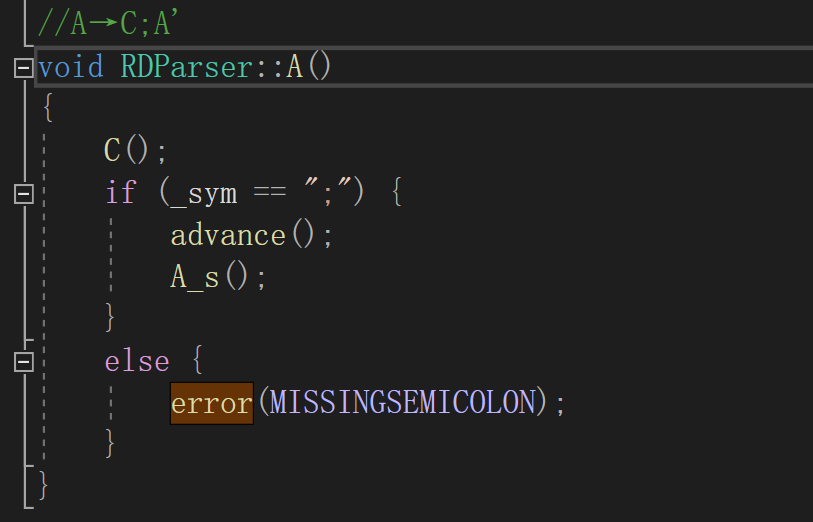
## 递归下降语法分析部分关键函数

### 递归子程序(状态转换图的实现)

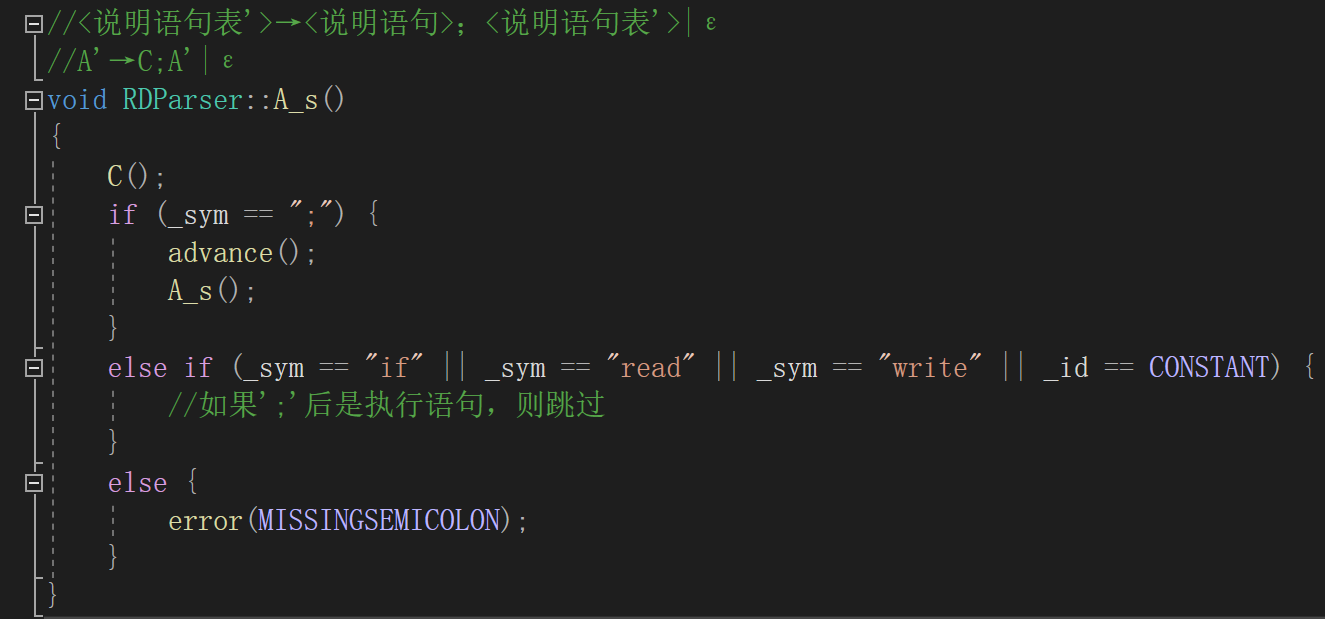
#### void s()

<程序>→begin <说明语句表><执行语句表> end

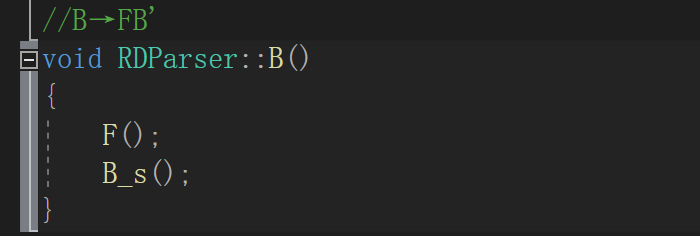
#### void A()

<说明语句表>→<说明语句>；<说明语句表'>

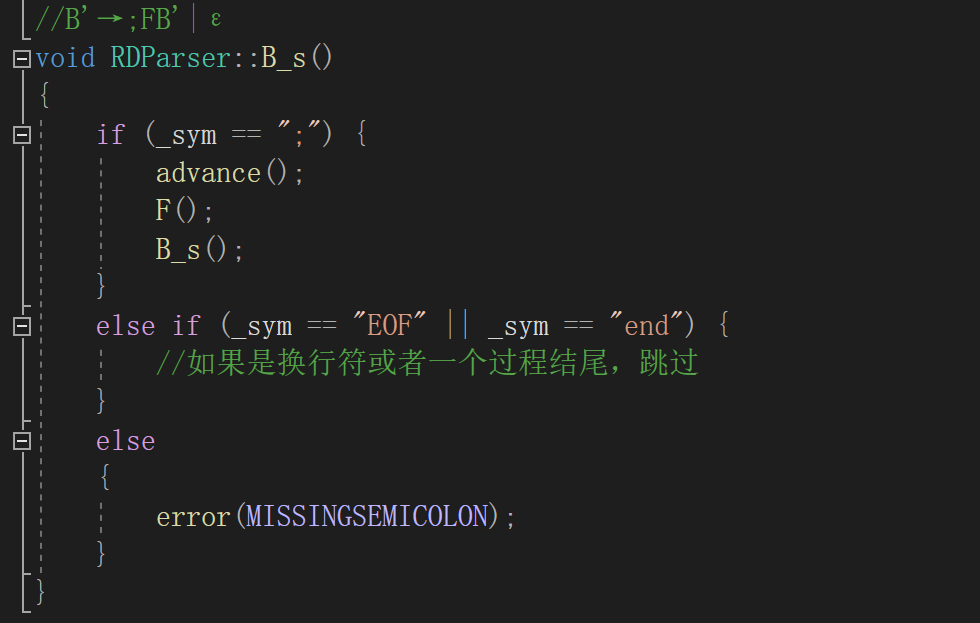
#### Void A\_s()

<说明语句表'>→<说明语句>；<说明语句表'>|ε

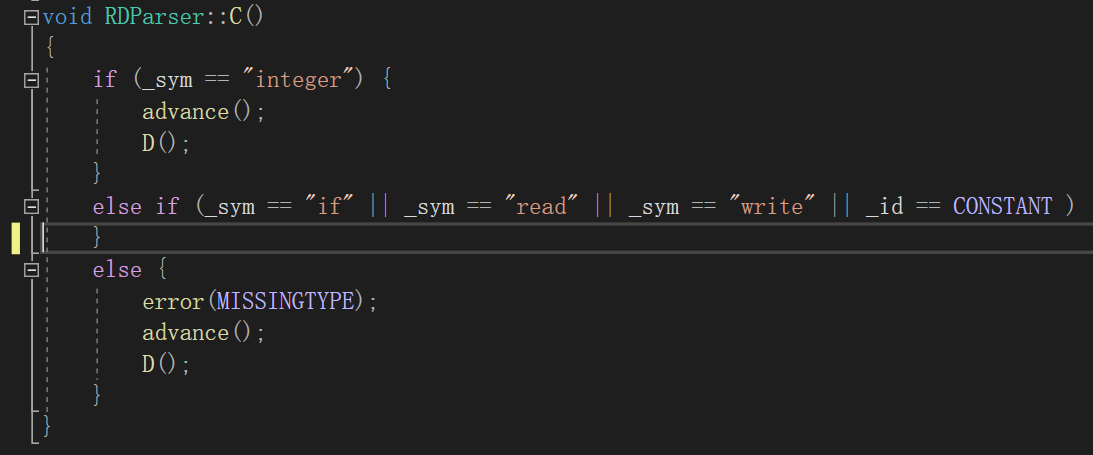
#### Void B()

<执行语句表>→<执行语句><执行语句表'>

#### Void B\_s()

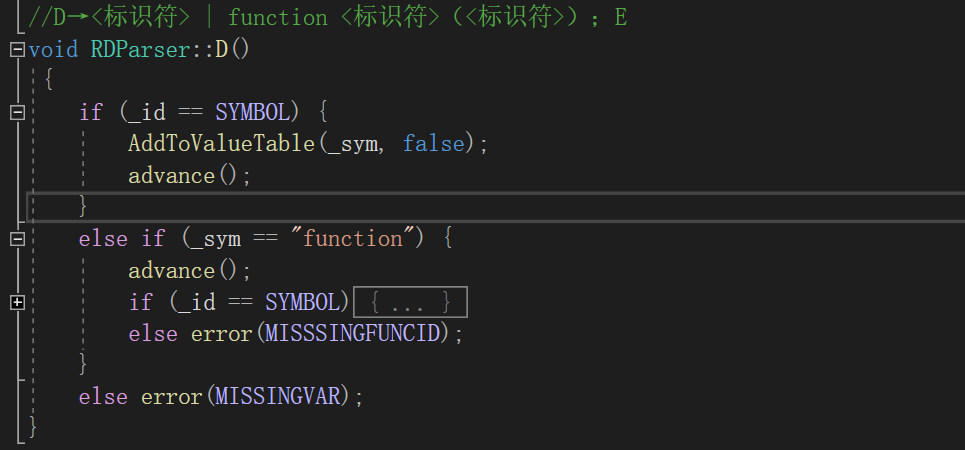
<执行语句表'>→; <执行语句><执行语句表'> | ε

#### Void c()

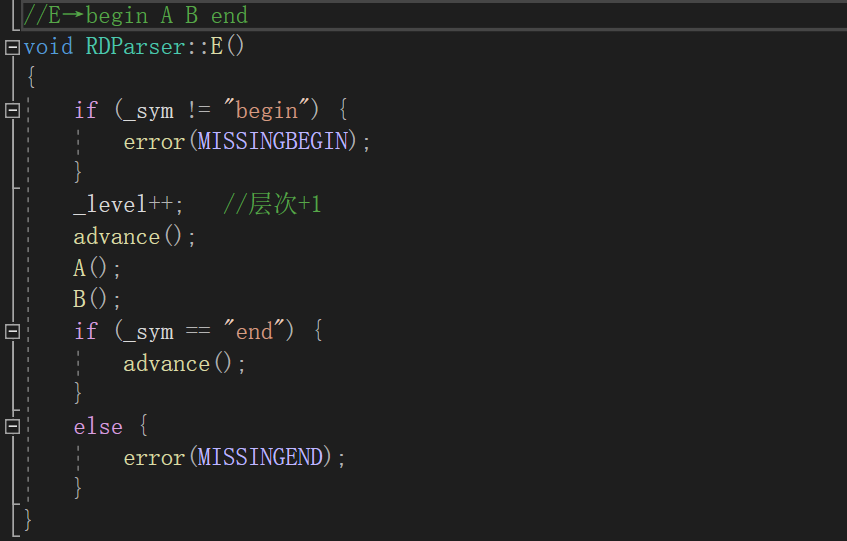
<说明语句>→integer <标识符或函数说明>

#### Void D()

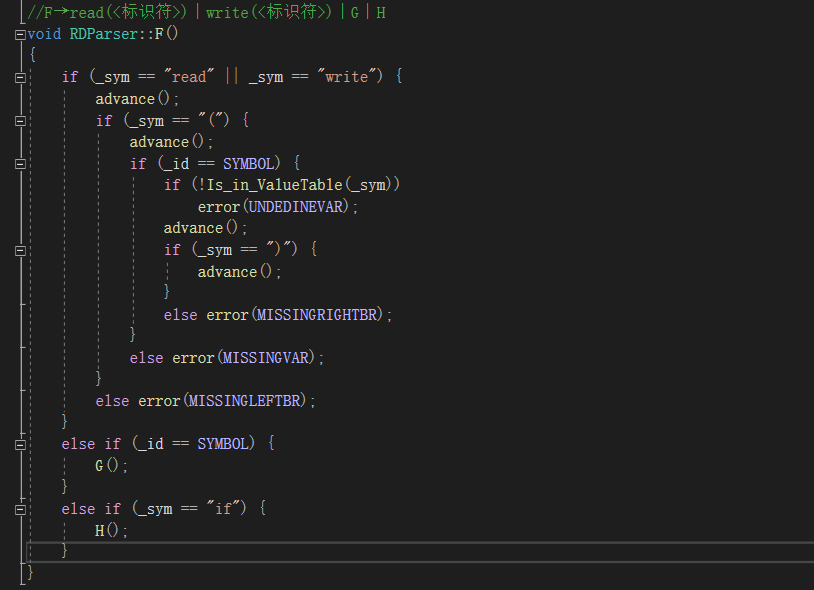
D→<标识符> | function <标识符>（<标识符>）；E

AddToValueTable（）：添加变量到变量名表中

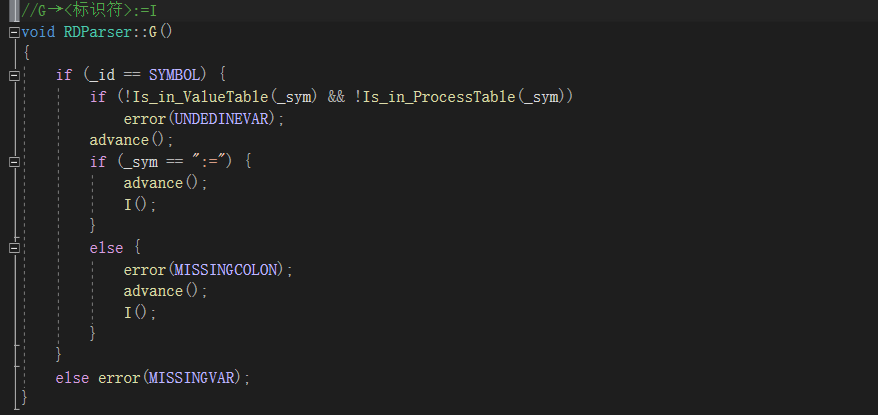
#### Void E()

<函数体>→begin <说明语句表><执行语句表> end

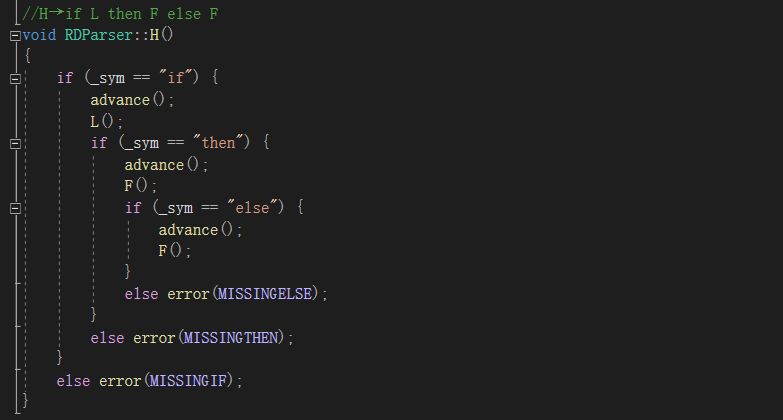
#### Void F()

<执行语句>→read(<标识符>)│write(<标识符>)│<赋值语句>│<条件语句>

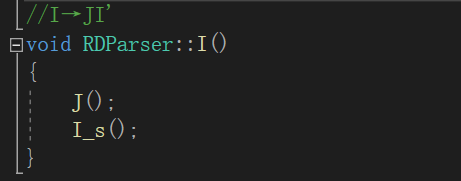
#### Void G()

<赋值语句>→<标识符>:=<算术表达式>

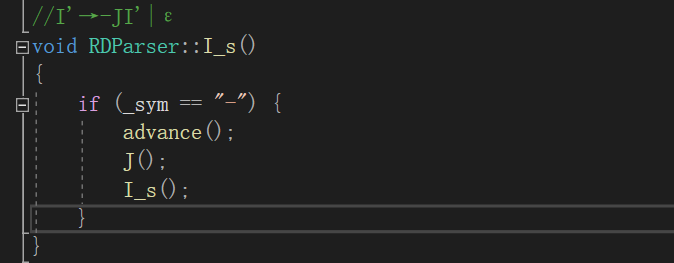
#### Void H()

<条件语句>→if<条件表达式>then<执行语句>else <执行语句>

#### Void I()

<算术表达式>→<项><算术表达式>'

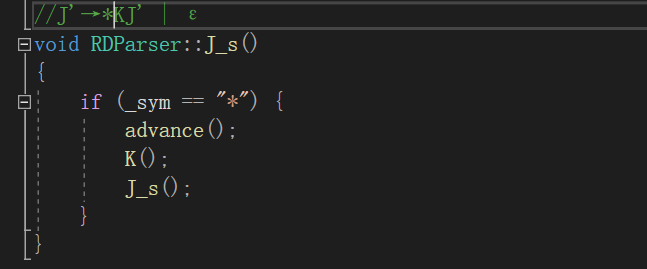
#### Void I\_s()

<算术表达式>'→-<项><算术表达式>'

#### Void J()

<项>→<因子><项'>

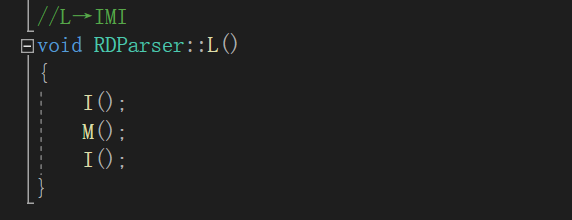
#### Void J\_s()

<项'>→\*<因子><项'> | ε

#### Void K()

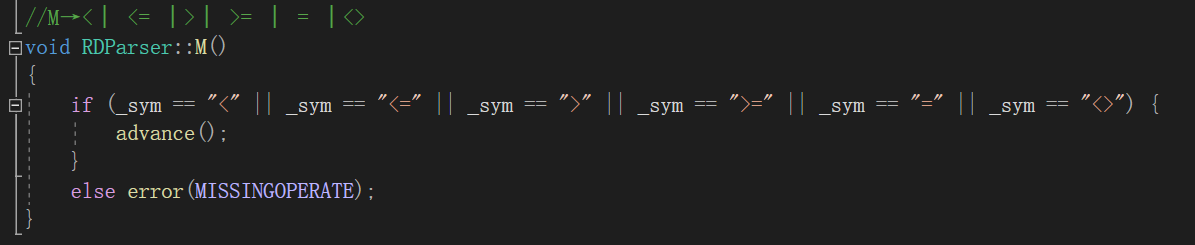
<因子>→<标识符>│<常数>│<函数调用>

#### Void L()

<条件表达式>→<算术表达式><关系运算符><算术表达式>

#### Void M()

<关系运算符> →<│<=│>│>=│=│<>



### 报错函数void RDParser::error(int type)

语法分析器根据传入类型报错，（展示部分代码

# 实验结果

求n！（该测试程序包含算术表达式，逻辑表达式，赋值语句，if-then分支结构，while循环结构）

## 正确测试用例

## 错误测试用例

### 错误文件error.err输出

### 词法分析result.dyd输出结果