

**编译原理**

# 数字控制器的设计

|  |  |  |
| --- | --- | --- |
| 姓名： | 贾浩男 | |
| 学号： | 16281007 | |
| 专业： | 计算机科学与技术 | |
|  |  |  |
|  |  |  |
| 提交日期： | 2019年 4月 9日 | |

## 程序功能描述

程序的主要功能为C语言的词法分析器，在为各个给定单词符号编码的基础上，可以根据输入文件in.txt中的代码内容输出二元式序列组成的中间文件，并且可以输出程序的错误词法提示信息，基本符合词法分析器的所要求功能。

### 单词编码

程序为包括保留字、用户定义标识符、运算符号等40类符号都进行了编码，其编码详情如下：

表格 1 编码详情

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **单词符号** | **类别编码** | **助记符** | **单词符号** | **类别编码** | **助记符** |
| void | 1 | 未设 | if | 2 | 未设 |
| else | 3 | for | 4 |
| while | 5 | int | 6 |
| float | 7 | double | 8 |
| return | 9 | 标识符 | 10 | ID |
| 整数 | 11 | INT | < | 12 | LT |
| <= | 13 | LE | == | 14 | EQ |
| != , <> | 15 | NE | > | 16 | GT |
| >= | 17 | GE | = | 18 | ASSIGN |
| + | 19 | ADD | ++ | 20 | ADD1 |
| - | 21 | SUB | -- | 22 | SUB1 |
| \* | 23 | MUL | / | 24 | DIV |
| /\* \*/, // | 25 | COMMENT | ; | 26 | SEMIC |
| ( | 27 | LB | ) | 28 | RB |
| ! | 29 | NOT | && | 30 | AND |
| || | 31 | OR | , | 32 | COMMA |
| { | 33 | LCB | } | 34 | RCB |
| += | 35 | ADDE | -= | 36 | SUBE |
| \*= | 37 | MULE | /= | 38 | DIVE |
| >> | 39 | MR | << | 40 | ML |

### 可识别正则文法

代码可识别的正则文法如下：

|  |
| --- |
| **<标识符>→字母 | <标识符>字母 | <标识符>数字**  **<无符号整数>→数字 | <无符号整数>数字**  **<单字符分界符>→+ | - | \* | / | ; | , | ( | ) |{ | } | < | > | = | > | < | ！**  **<双字符分界符>→<大于>= | <大于>> |<小于>= | <小于>> | <小于> < | <等于>= | <除号>\* | <加号>+ | <减号>- | <加号>= | <减号>= | <乘号>= | <除号>= | <除号>/ | <或>| | <且>&/ | <非>=**  **<小于>→<**  **<等于>→=**  **<大于>→>**  **<加号>→+**  **<减号>→-**  **<乘号>→\***  **<除号>→/**  **<或>→|**  **<且>→&**  **<非>→！**  **该语言的保留字 :void、int、float、double、if、else、for、while、return**  **用户定义标识符ID的限制：不超过32位且大小写敏感** |

## 程序设计

### 函数列支

表格 2 总程序列支

|  |  |  |  |
| --- | --- | --- | --- |
| **函数名称** | **参数** | **返回值** | **功能描述** |
| init\_map() | -- | void | 将所编码的保留字加载入map<int, string> reserved所定义的map映射；  将所编码的剩余符号加载入map<int, string> word所定义的map映射； |
| lookup() | **char** \* tok | int | 查找token是否在保留字中，若在返回类别编码；若不在，返回0。 |
| out() | int c,  string tok | void | 根据传入符号编码c、  传入字符串tok决定输出内容。 |
| report\_error() | int line,  string tips | void | 根据输入行数line定位出错地点，根据出错信息tips输出报错警告内容。 |
| scanner() | FILE \*fp | void | 根据输入文件扫描代码  从而实现词法分析器功能 |

### 函数调用流程

根据上一节中所定义的各个函数的功能及输入输出，可构造程序中的整体函数调用流程如下图所示：

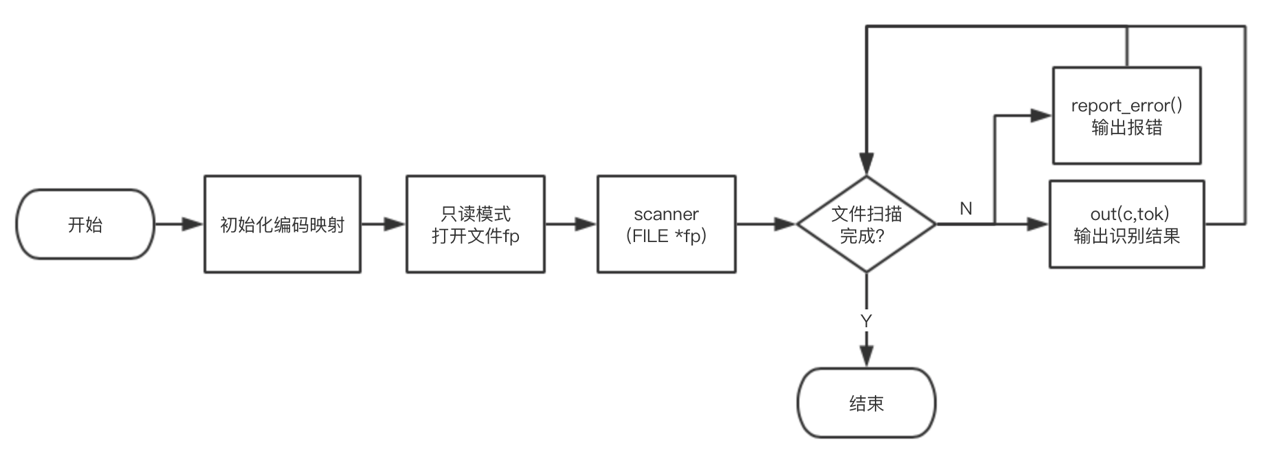


图 1 函数调用流图

### scanner函数实现

有了函数定义与总的调用流程之后，下面来讲述子函数的具体实现流程。

子函数中out和report\_error函数为map中的匹配输出、lookup为map中的匹配查找，实现比较容易，故此处不做赘述。

接下来主要讲述词法分析的扫描程序scanner函数的实现细节：



图 2 scanner函数流程

上图即为scanner函数的主要实现细节，鉴于篇幅的限制，图中省去了两个逻辑运算符（&&，||）的处理流程，不过具体实现细节与其余的双字符分界符的实现相仿。

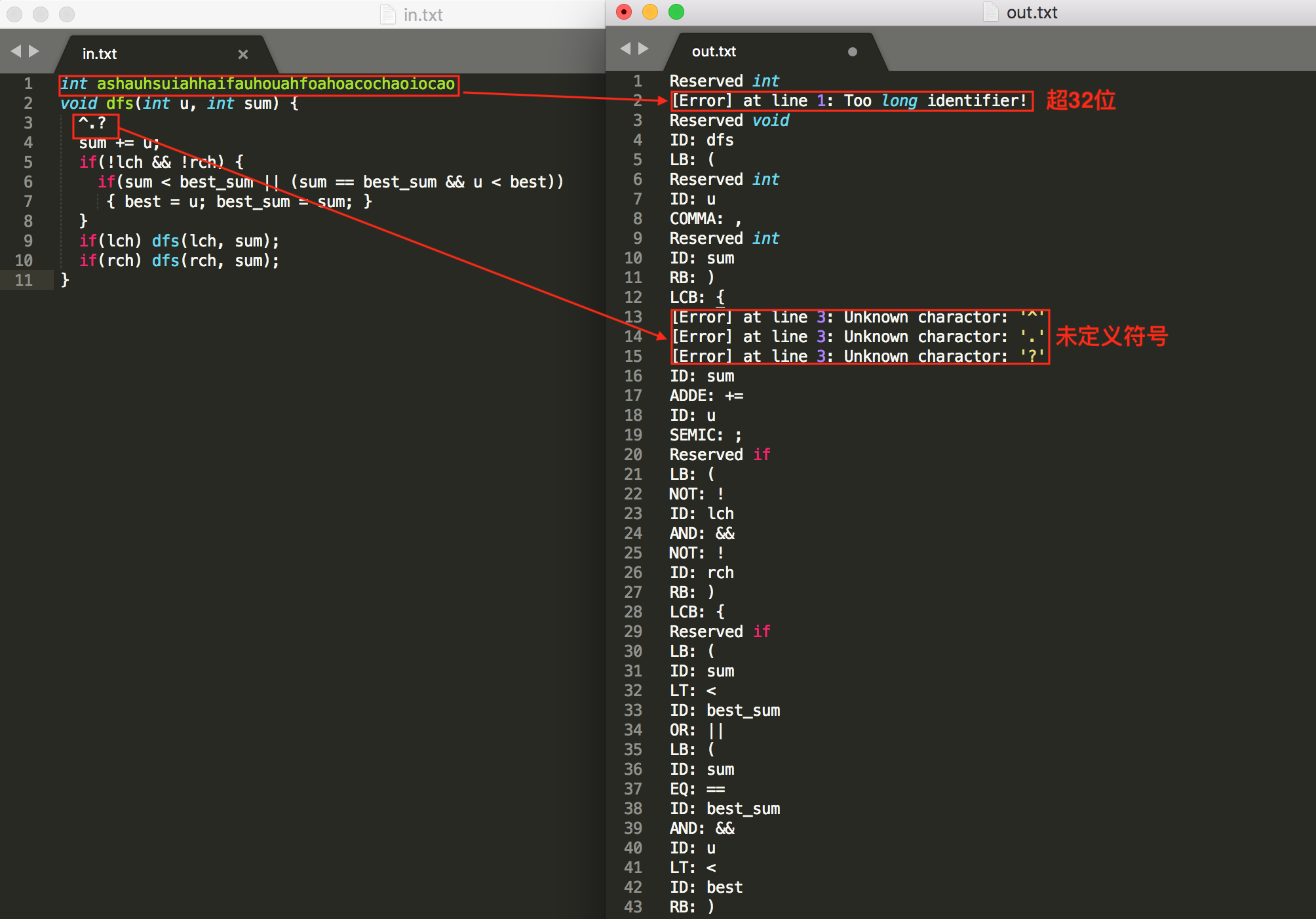
可以看到，该函数的主体分为四种字符的判别，分别是——**用户定义标识符**、**整数**、**单字符分界符**和**双字符分界符**。

* **用户定义标识符：**用户定义标识符需要判断是否以字母开头，在确定以字母开头后循环向后扫描，扫描字符包括字母、数字和下划线不等，直到扫描到第一个不合法字符为止。在扫描结束后需要首先判断长度是否已经超出32位限制，若超出限制则调用report\_error函数报错，否则则调用lookup函数在保留字映射中寻找是否匹配，如果匹配则为保留字定义，否则则为合法的用户定义标识符。
* **整数：**整数的判断为：当字符碰到数字后，循环判断其后继字符是否仍然为数字，直到扫描到不为数字的位置停止。拼接在token中的字符串即为该整数的表示。
* **单字符分界符**：单字符的判断方法较为容易，利用switch-case结构独立判断即可。
* **双字符分界符**：双字符分界符建立在有些单字符的基础之上，在判断时需要在检索到第一个匹配项的基础上继续使fp指针向下走一步，如果与双字符分界符匹配则调用out输出，否则指针回退一步，匹配单字符。

## 代码测试

一切就绪之后，下面选用两组文件进行代码测试，测试集基本涵盖说实现功能，为了使结果更直观，第一维输出均取助记符，且用Reserved标识。

### 测试一



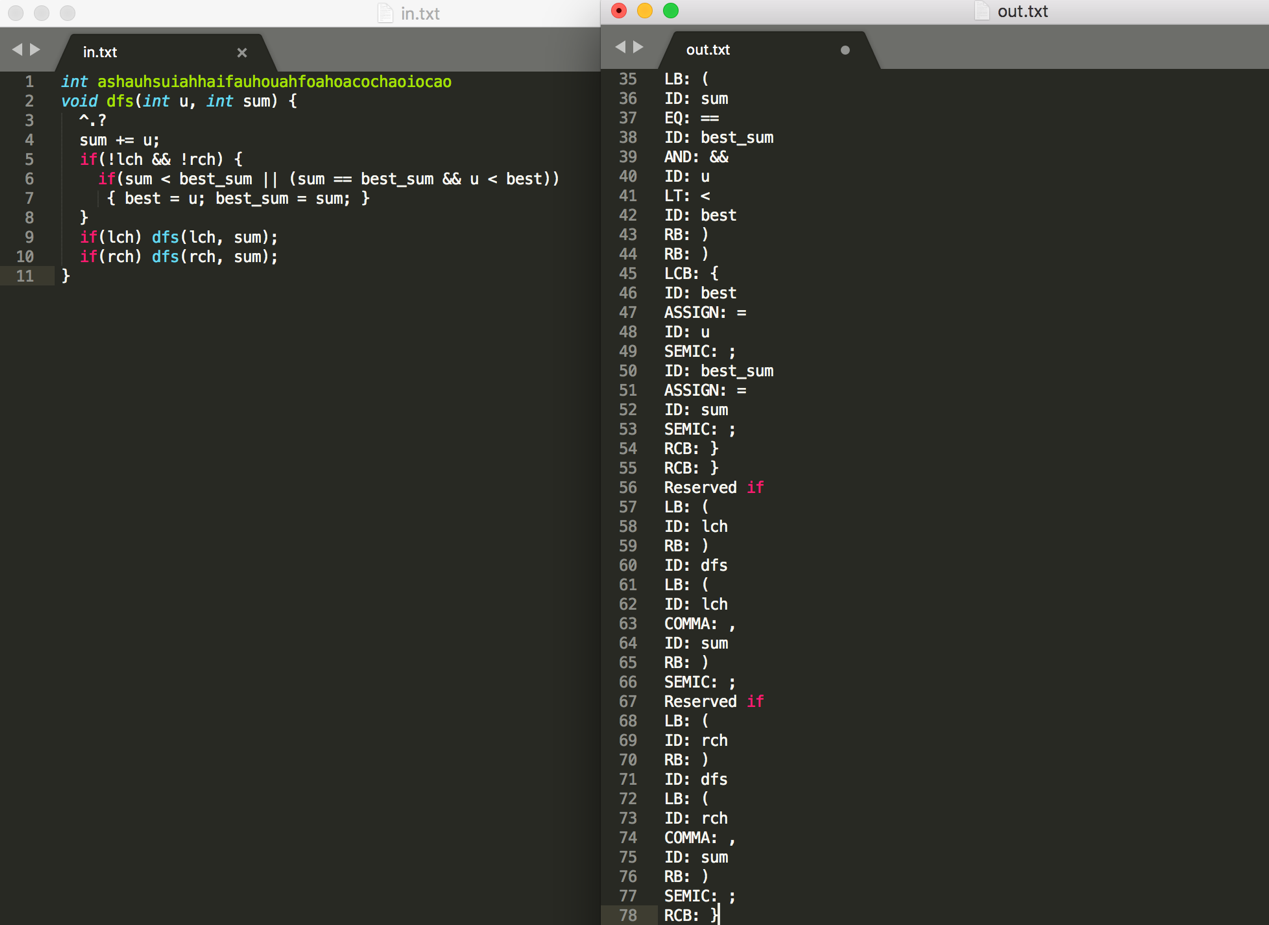
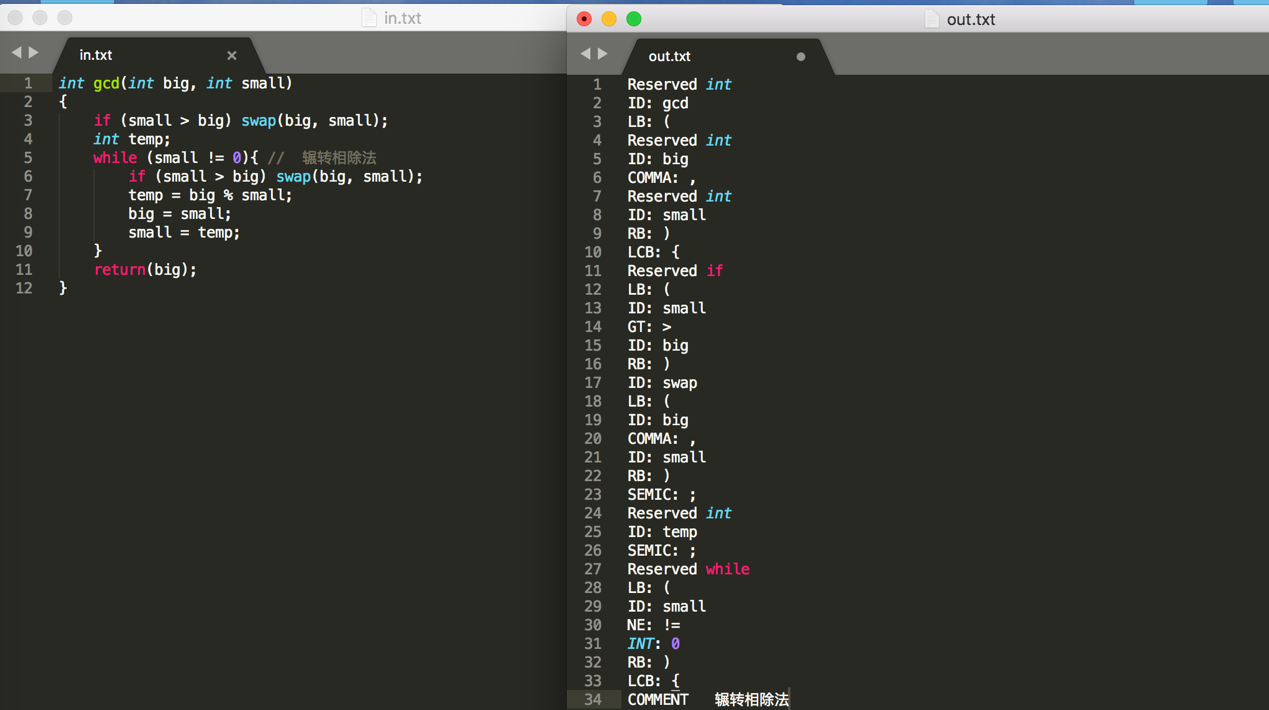


图 3 测试一输出

除正确匹配符号样例外，报错上，测试一主要测试了未定义符号以及超长定义的错误样例。

### 测试二

除正确匹配符号样例外，在报错上，测试二主要测试了不封闭注释的错误，具体如下：



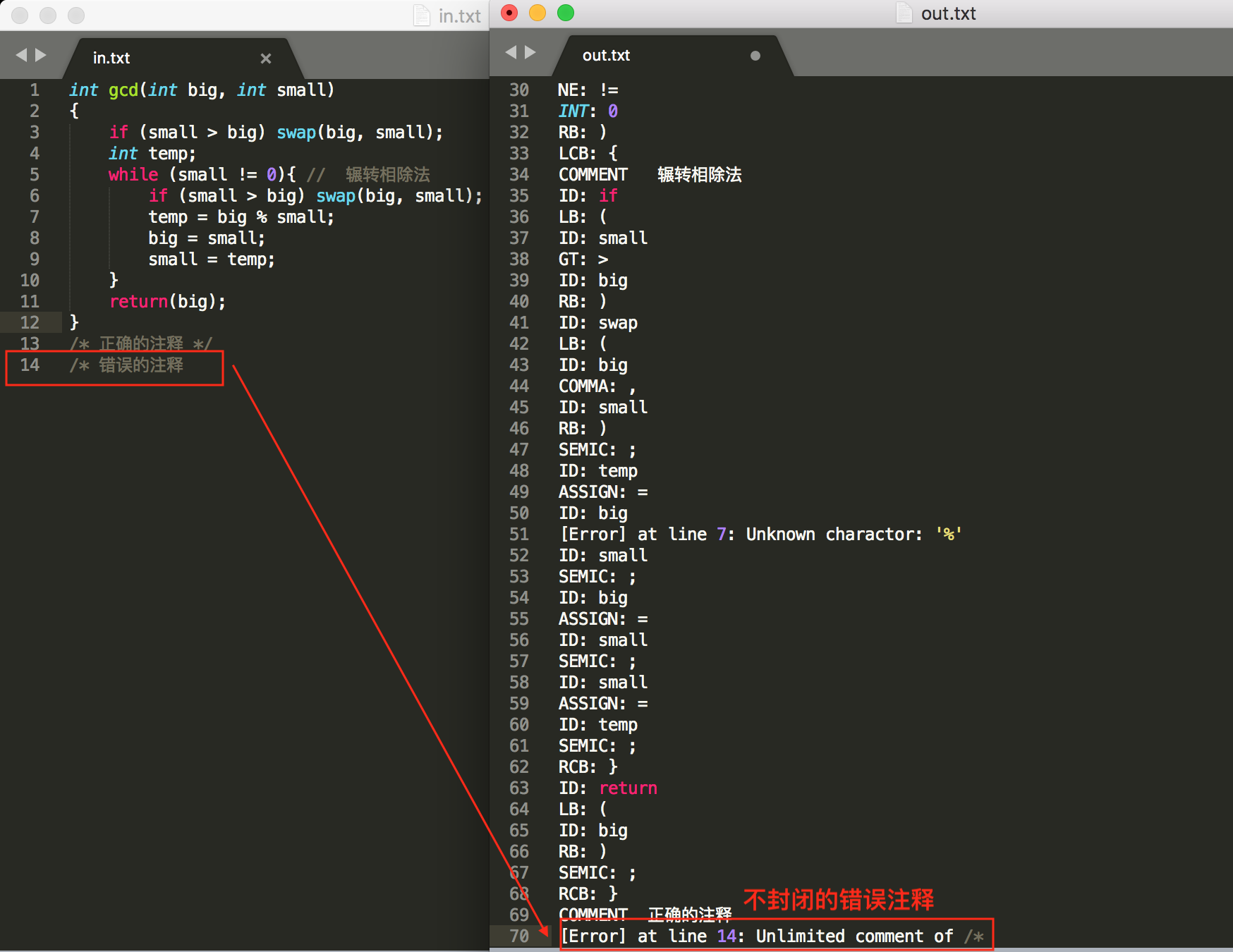


图 4 测试二输出

可以看到，两个测试样例基本涵盖所有内容，程序可以识别所有已预定义好的符号。