编译小作业1.2——Toy语言的词法分析

在编译小作业1中，你需要对某个简单的编程语言（称为Toy语言）进行分析，并自己编写代码，实现词法分析和语法分析。

编译小作业1分为不同部分，将逐步公布在网络学堂作业区。如对作业内容有疑义或问题，可以联系助教。

Toy语言的介绍请参考编译小作业1.1的作业附件。

在编译小作业1.2中，会给出完整的语法规则与词法规则。你需要基于写出一个对应的词法分析代码。

语法规则与词法规则定义

给定如下语法规则 `G[PROGRAM]` （其中，大写符号表示非终结符，小写符号和引号(单引号或双引号)引起来的部分表示终结符）：

```
PROGRAM          -> STATEMENT PROGRAM | ϵ
STATEMENT        -> INPUT_STMT | PRINT_STMT | DECLARATION_STMT | ASSIGNMENT_STMT
| IF_STMT | WHILE_STMT
INPUT_STMT       -> "input" identifier ";"
PRINT_STMT       -> "print" identifier ";" | "print" string_literal ";"
DECLARATION_STMT -> "var" identifier ";"
ASSIGNMENT_STMT  -> identifier "=" EXPRESSION ";"
IF_STMT          -> "if" "(" CONDITION ")" "{" PROGRAM "}"
WHILE_STMT       -> "while" "(" CONDITION ")" "{" PROGRAM "}"

EXPRESSION       -> TERM | TERM "+" TERM | TERM "-" TERM
TERM             -> FACTOR | FACTOR "*" FACTOR | FACTOR "/" FACTOR
FACTOR           -> identifier | number | "(" EXPRESSION ")"

CONDITION        -> EXPRESSION "==" EXPRESSION | EXPRESSION ">" EXPRESSION |
                    EXPRESSION "<" EXPRESSION
```

对应的词法规则为：

```
identifier       -> letter identifier_rest
identifier_rest  -> letter identifier_rest | digit identifier_rest | ϵ
number           -> digit number_rest | "-" digit number_rest
number_rest      -> digit number_rest | ϵ
string_literal   -> '"' characters '"'
characters       -> character characters | ϵ
character        -> letter | digit | special_char
special_char     -> " " | "!" | "#" | "$" | "%" | "&" | "'" | "(" | ")" | "*" |
"+" | "," | "-" | "." | "/" | ":" | ";" | "<" | "=" | ">" | "?" | "@" | "[" | "\"
| "]" | "^" | "_" | "{" | "|" | "}" | "~"

letter           -> "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" | "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"
digit            -> "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

作业任务——词法分析

编译小作业1共15分，其中本次作业占4分。需要实现一个Toy语言的词法分析器（**不能直接使用词法分析工具，需要自己编写代码实现**）。

词法分析器需要将给定的Toy语言源代码作为输入，输出分析得到的token流（包括token类型名称与token的内容）。

- token类型有多种，包括：
  - 词法规则中定义的`identifier`、`number`、`string_literal`
  - 语法规则中的其他终结符（如`var`、`if`、`while`、`input`、`print`、`=`、`(`等）。
    - 这些终结符需要你自己给他起名字作为所谓的“token类型名称”。比如你可以将`(`叫做`lparen`，更多例子可以见后方示例中的打印出的参考结果，但你不一定要和后面的保持完全一致，命名合理即可。

请**实现一个词法分析程序，该程序读取源代码文件后，输出token流。**

提示：

- 可以从左到右扫描输入的源代码文件内容，进行正则表达式匹配。但**不能直接用正则表达式进行全局匹配，只能从剩余输入串的开头进行匹配。**以避免出现识别错误的情况（比如：将字符串中的`if`识别为条件语句中的`if`关键字）。
- 请注意，在之前的词法规则定义中，special_char中没有双引号字符。因此可以直接用正则识别字符串的起始与结束位置。
- 除非存在于字符串内部，所有空白字符（空格、换行符、制表符等）可以被理解为分隔符，词法分析过程直接跳过即可。
  - 但需注意，终结符中不会有额外的空白字符。例如，`3 1`会被理解成两个`number`（`3`和`1`），而非一个`number`（`31`）。
- `identifier`、`number`、`string_literal`可以直接作为单个token进行处理，针对他们分别设计正则表达式。

示例：

- 对于以下Toy语言的源代码：

```
var x;
var y;
input x;
if (x > 5) {
    y = x * (x / 2 + 10);
}
print "After if, finished!";
```

- 打印出的参考结果为：

```
('var', 'var')
('identifier', 'x')
('semicolon', ';')
('var', 'var')
('identifier', 'y')
('semicolon', ';')
('input', 'input')
('identifier', 'x')
('semicolon', ';')
('if', 'if')
('lparen', '(')
('identifier', 'x')
('greater', '>')
('number', '5')
('rparen', ')')
('lbrace', '{')
('identifier', 'y')
('assign', '=')
('identifier', 'x')
('multiply', '*')
('lparen', '(')
('identifier', 'x')
('divide', '/')
('number', '2')
('plus', '+')
('number', '10')
('rparen', ')')
('semicolon', ';')
('rbrace', '}')
('print', 'print')
('string_literal', '"After if, finished!"')
('semicolon', ';')
```

提交作业时，请提交一个压缩包，包括源代码文件、可执行文件和一个文档，文档中展示三个你自己设计的测试源代码以及对应输出结果。