Lab1 词法实验

正则表达式

正则表达式（Regex）是描述文本模式的工具，用于字符串匹配、搜索和替换。它由字符和特殊符号组成，形成匹配文本的模式。正则表达式在编程和文本处理中非常强大。

下面是计算机程序中一些常见的正则表达式符号及其含义：

符号	含义
`.`	匹配除换行符以外的任何字符
`*`	匹配前一个字符的零次或多次出现
`+`	匹配前一个字符的一次或多次出现
`?`	匹配前一个字符的零次或一次出现
`[]`	定义一个字符类，匹配括号内的任何一个字符
`[^]`	否定字符类，匹配不在括号内的任何一个字符
`()`	创建一个子表达式，并捕获匹配的部分
|	或操作，匹配两边任一表达式
`{n}`	匹配前一个字符恰好 n 次
`{n,}`	匹配前一个字符至少 n 次
`{n,m}`	匹配前一个字符至少 n 次，但不超过 m 次
`\`	转义字符，用于匹配特殊字符
`^`	锚点，匹配字符串的开头
`$`	锚点，匹配字符串的结尾
`\d`	匹配任何数字字符
`\D`	匹配任何非数字字符
`\w`	匹配任何字母、数字或下划线字符
`\W`	匹配任何非字母、非数字、非下划线字符
`\s`	匹配任何空白字符（空格、制表符、换行等）

符号	含义
\S	匹配任何非空白字符

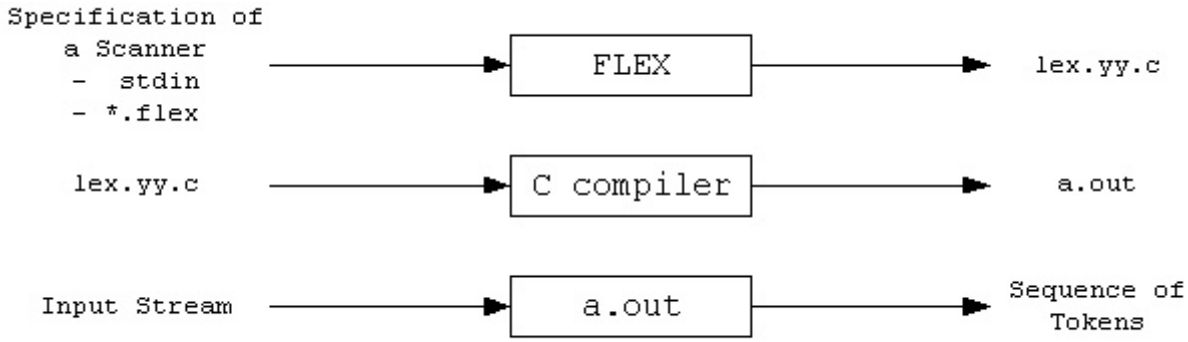
这些符号允许你创建复杂的 pattern，用于匹配和操作文本中的字符串。不同的编程语言和工具可能会略有不同，但基本的正则表达式符号通常是通用的。

这是一个正则表达式在线匹配网站（https://c.runoob.com/front-end/854/），你可以在该网站创建相关正则 pattern 进行匹配，以熟悉计算机程序中的正则表达式符号。

flex

简介

Flex 执行过程如下：



1. 首先，Flex 源程序中的规则被转换成状态转换图，生成对应的代码，包括核心的 yylex() 函数，保存在 lex.yy.c 文件中。Flex 源程序通常以 .l 为后缀，按照 Flex 语法编写，用于描述词法分析器。
2. 生成的 lex.yy.c 文件可以通过 C 编译为可执行文件。
3. 最终，可执行文件将输入流解析成一系列的标记（tokens）。

使用说明

Flex源代码文件包括三个部分，由“%%”隔开，如下所示：

```
1 声明部分
2 %%
3 规则部分
4 %%
5 C代码部分
```

- 声明部分包含名称声明和选项设置，%{ 和 %} 之间的内容会被原样复制到生成的 C 文件头部，可用于编写 C 代码，如头文件声明和变量定义等。
- 规则部分位于两个 %% 之间，包括多条规则，每个规则由正则表达式定义的模式和与之匹配的 C 代码动作组成。当词法分析程序识别出某模式时，执行相应的 C 代码。

- C 代码部分可包括 main() 函数，用于调用 yylex() 执行词法分析。yylex() 是由 Flex 生成的词法分析例程，默认从 stdin 读取输入文本。

我们结合具体的例子，来看Flex的源程序的三部分结构：

```
/* %option noyywrap 功能较为复杂，同学们自行了解 */
%option noyywrap
%{
/* Flex 源程序采样类 C 的语法和规则 */
/* 以下是声明部分，`%{` 和 `%}` 之间的内容会被原样复制到生成的 C 文件头部
    包括该条注释内容 */
#include <string.h>
int chars = 0;
int words = 0;
%}

/* 以下是规则部分，在规则部分写注释不能顶格写 */
/* 每条规则由正则表达式和动作组成 */
/* 第一条规则匹配纯字母的字符串，并统计字母个数和字符串个数
                        其中 yytext 为匹配到的 token */
/* 第二条规则匹配其他字符或字符串并执行空动作 */
%%
 /* 在规则部分，不要顶格写注释 */
[a-zA-Z]+  { chars += strlen(yytext); words++; }
.          {}
%%

/* 以下为 C 代码部分 */
int main()
{
    /* yylex() 是由 Flex 自行生成的，用于执行 */
    // yyin = fopen(buffer,"r"); 设定yylex从buffer所指文件读取内容
    yylex();
    /* 对于 stdin 输入匹配结束，执行其他操作 */
    printf("look, I find %d words of %d chars\n", words, chars);
    return 0;
}
```

运行测试

编写完程序之后，使用如下指令运行flex：

```
flex PATH_TO_FLEX_CODE
```

其中PATH_TO_FLEX_CODE为编写的flex程序的路径。

运行结束之后，会生成lex.yy.c文件。编译并运行该文件即可进行测试。

```
gcc lex.yy.c -o a
./a
```

实验要求

借助flex工具实现一个词法分析器，识别出SysY代码中的所有终结符，然后按照单词符号出现顺序依次输出：原始单词符号、该符号出现的行数、该符号出现的列数起始与终止位置。

例如输入：

```
void main(void) { return; }
```

则应该输出：

```
Text      Line    Column (Start,End)    Type
void      1       (1,5)
main      1       (6,10)                Ident
(         1       (10,11)
void      1       (11,15)
)         1       (15,16)
{         1       (17,18)
return    1       (19,25)
;         1       (25,26)
}         1       (27,28)
```

特别提示

以下列举几个编写规则时需要注意的重难点：

- 对于注释的处理，包括单行注释和成段注释。
- windows和linux下换行符的差别。
- 包含数字的变量和数字之间的区分。
- 八进制、十进制、十六进制数的处理。
- 建议实现错误处理。