Lex 使用指南

Lex 是由美国 Bell 实验室 M.Lesk 等人用 C 语言开发的一种词法分析器自动生成工具，它提供一种供开发者编写词法规则（正规式等）的语言（Lex 语言）以及这种语言的翻译器（这种翻译器将 Lex 语言编写的规则翻译成为 C 语言程序）。

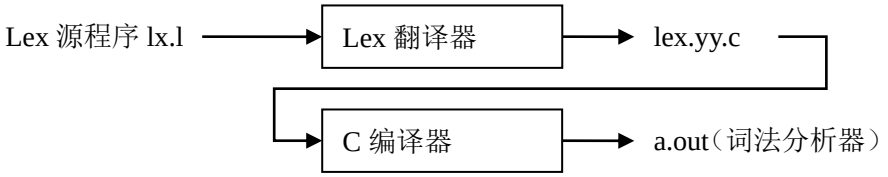
Lex 是 linux 下的工具，本实验使用的编译工具是 Flex（the Fast Lexical Analyzer Generator），它与 lex 的使用方法基本相同，只有很少的差别。编译环境为 linux 环境，或者在 windows 下使用模拟 linux 环境的工具（例如 cygwin）模拟 linux 下的编译环境。

一、Lex 的基本原理和使用方法

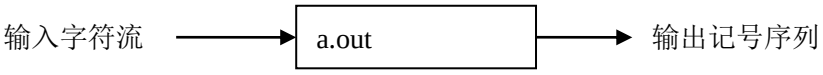
Lex 的基本工作原理为：由正规式生成 NFA，将 NFA 变换成 DFA，DFA 经化简后，模拟生成词法分析器。

其中正规式由开发者使用 Lex 语言编写，其余部分由 Lex 翻译器完成.翻译器将 Lex 源程序翻译成一个名为 lex.yy.c 的 C 语言源文件，此文件含有两部分内容：一部分是根据正规式所构造的 DFA 状态转移表，另一部分是用来驱动该表的总控程序 yylex()。当主程序需要从输入字符流中识别一个记号时，只需要调用一次 yylex()就可以了。为了使用 Lex 所生成的词法分析器，我们需要将 lex.yy.c 程序用 C 编译器进行编译，并将相关支持库函数连入目标代码。Lex 的使用步骤可如下图所示：

生成词法分析器的过程：



使用所生成的词法分析器的过程：



二、lex 源程序的写法：

Lex 源程序必须按照 Lex 语言的规范来写，其核心是一组词法规则（正规式）。一般而言，一个 Lex 源程序分为三部分，三部分之间以符号%%分隔。

[第一部分：定义段]
%%
第二部分：词法规则段
[%%
第三部分：辅助函数段]

其中，第一部分及第三部分和第三部分之上的%%都可以省略（即上述方括号括起的部

分可以省略）。以%开头的符号和关键字，或者是词法规则段的各个规则一般顶着行首来写，前面没有空格。

Lex 源程序中可以有注释，注释由/*和*/括起，但是请注意，注释的行首需要有前导空白。

1. 第一部分定义段的写法：

定义段可以分为两部分：

第一部分以符号%{和%}包裹，里面为以 C 语法写的一些定义和声明：例如，文件包含，宏定义，常数定义，全局变量及外部变量定义，函数声明等。这一部分被 Lex 翻译器处理后会全部拷贝到文件 lex.yy.c 中。注意，特殊括号%{和%}都必须顶着行首写。例如：

```
%{
  #define LT 1
  int yylval;
%}
```

第二部分是一组正规定义和状态定义。正规定义是为了简化后面的词法规则而给部分正规式定义了名字。每条正规定义也都要顶着行首写。例如下面这组正规定义分别定义了 letter，digit 和 id 所表示的正规式：

```
letter      [A-Za-z]
digit       [0-9]
id          {letter}({letter}|{digit})*
```

注意：上面正规定义中出现的小括号表示分组，而不是被匹配的字符。而大括号括起的部分表示正规定义名。**正规式右侧的表达式里不要出现多余空格，否则可能会编译出错。**

状态定义也叫环境定义，它定义了匹配正规式时所处的状态的名字。状态定义以%s 开始，后跟所定义的状态的名字，注意%s 也要顶行首写，例如下面一行就定义了一个名为 COMMENT 的状态和一个名为 BAD 的状态，状态名之间用空白分隔：

```
%s COMMENT BAD
```

2. 第二部分词法规则段的写法：

词法规则段列出的是词法分析器需要匹配的正规式，以及匹配该正规式后需要进行的相关动作。其例子如下：

```
while           {return (WHILE);}
do              {return (DO);}
{id}            {yylval = installID (); return (ID);}
```

每行都是一条规则，该规则的前一部分是正规式，需要顶行首写，后一部分是匹配该正规式后需要进行的动作，这个动作是用 C 语法来写的，被包裹在{}之内，被 Lex 翻译器翻译后会被直接拷贝进 lex.yy.c。正规式和语义动作之间要有空白隔开。其中用{}扩住的正规式表示正规定义的名字。

也可以若干个正规式匹配同一条语义动作，此时正规式之间要用 | 分隔。

3. 第三部分辅助函数段的写法：

辅助函数段用 C 语言语法来写，辅助函数一般是在词法规则段中用到的函数。这一部分一般会被直接拷贝到 lex.yy.c 中。

4. Lex 源程序中词法规则（即正规式）的相关规定：

元字符：元字符是 lex 语言中作特殊用途的一些字符，包括：* + ? | { } [ ] ( ) . ^ $ " \ - / < >。

正文字符：除元字符以外的其他字符，这些字符在正规式中可以被匹配。若单个正文字符 c 作为正规式，则可与字符 c 匹配，元字符无法被匹配，如果元字符想要被匹配，则需要通过“转义”的方式，即用" "包括住元字符，或在元字符前加\。例如"+"和\+都表示加号。C 语言中的一些转义字符也可以出现在正规式中，例如\t \n \b 等。

部分元字符在 lex 语言中的特殊含义：

^表示补集：[^…]表示补集，即匹配除^之后所列字符以外的任何字符。如[^0-9]表示匹配除数字字符 0-9 以外的任意字符。除^ - \以外，任何元字符在方括号内失去其特殊含义。如果要在方括号内表示负号-，则要将其至于方括号内的第一个字符位置或者最后一个字符位置，例如[-+0-9] [+0-9-]都匹配数字及+ - 号。

. ^ $ /：

点运算符 . 匹配除换行之外的任何字符，一般可作为最后一条翻译规则。

^匹配行首字符。如：^begin 匹配出现在行首的 begin

$匹配行末字符。如：end$ 匹配出现在行末的 end

R1/R2（R1 和 R2 是正规式）表示超前搜索：若要匹配 R1，则必须先看紧跟其后的超前搜索部分是否与 R2 匹配。

如：DO/{alnum}*={alnum}*,表示如果想匹配 DO，则必须先在 DO 后面找到形式为{alnum}*={alnum}*,的串，才能确定匹配 DO。

5. Lex 源程序中常用到的变量及函数：

yyin 和 yyout：这是 Lex 中本身已定义的输入和输出文件指针。这两个变量指明了 lex 生成的词法分析器从哪里获得输入和输出到哪里。默认：键盘输入，屏幕输出。

yytext 和 yyleng：这也是 lex 中已定义的变量，直接用就可以了。

yytext：指向当前识别的词法单元（词文）的指针

yyleng：当前词法单元的长度。

yyless(n)和 yymore()：这是 lex 中已定义的函数，直接用就可以了。

yyless(n)：yyless（n）将当前词法单元的除前 n 个字符外的所有字符返回给输入流（回退）

yymore：告诉扫描程序下次匹配规则时，应将相应的词法单元附加到 yytext 的当前值上，而不是替换它。

ECHO：Lex 中预定义的宏，可以出现在动作中，相当于 fprintf(yyout, “%s”,yytext)，即输

出当前匹配的词法单元。

yylex()：词法分析器驱动程序，用 Lex 翻译器生成的 lex.yy.c 内必然含有这个函数。

yywrap()：词法分析器遇到文件结尾时会调用 yywrap()来决定下一步怎么做：
若 yywrap()返回 0，则继续扫描
若返回 1，则返回报告文件结尾的 0 标记。
由于词法分析器总会调用 yywrap，因此辅助函数中最好提供 yywrap，如果不提供，则在用 C 编译器编译 lex.yy.c 时，需要链接相应的库，库中会给出标准的 yywrap 函数（标准函数返回 1）。

6. 词法分析器的状态（环境）：

词法分析器在匹配正规式时，可以在不同状态（或环境）下进行。我们可以规定在不同的状态下有不同的匹配方式。每个词法分析器都至少有一个状态，这个状态叫做初始状态，可以用 INITIAL 或 0 来表示，如果还需要使用其他状态，可以在定义段用%s 来定义。

使用状态时，可以用如下方式写词法规则：

```
<state1, state2> p0     {action0;}
<state1> p1             {action1;}
```

这两行词法规则表示：在状态 state1 和 state2 下，匹配正规式 p0 后执行动作 action0，而只有在状态 state1 下，才可以匹配正规式 p1 后执行动作 action1。如果不指明状态，默认情况下处于初始状态 INITIAL。

要想进入某个特定状态，可以在动作中写上这样一句： BEGIN state; 执行这个动作后，就进入状态 state。

下面是一段处理 C 语言注释的例子，里面用到了状态的转换，在这个例子里，使用不同的状态，可以让词法分析器在处于注释中和处于注释外时使用不同的匹配规则：

```
…
%s c_comment
…
%%
<INITIAL>"/*"       {BEGIN c_comment;}
…
<c_comment>"*/"     {BEGIN 0;}
<c_comment>.        {;}
```

7. Lex 的匹配策略：

1. 按最长匹配原则确定被选中的单词
2. 如果一个字符串能被若干正规式匹配，则先匹配排在前面的正规式。

三、Lex 生成的词法分析器如何使用：

lex 常常与语法分析器的生成工具 yacc（课本第三章会讲到）同时使用。此时，一般来说，语法分析器每次都调用一次 yylex()获取一个记号。如果想自己写一个程序使用 lex 生成

的词法分析器，则只需要在自己的程序中按需要调用 yylex()函数即可。

请注意：yylex()调用结束后，输入缓冲区并不会被重置，而是仍然停留在刚才读到的地方。并且，词法分析器当前所处的状态（%s 定义的那些状态）也不会改变。

完整的 Lex 源程序例子请见附件 exam1.l 和 exam2.l。

四、编译链接 lex 源程序的命令：

以下所有命令都为 linux 控制台下的命令，且假设当前处于工作目录下，所有文件都存在于当前目录下。

1. 用 lex 翻译器编译 lex 源程序命令（假设 filename.l 是 lex 源程序名，该文件在当前目录下）： flex filename.l
2. 用 gcc 编译器编译 lex 翻译器生成的 c 源程序（lex 翻译器生成的 c 源程序名固定为 lex.yy.c）： gcc [-o outfile] lex.yy.c –lfl

   其中，-lfl 是链接 flex 的库函数的，库函数中可能包含类似 yywrap 一类的标准函数。-o outfile 是可选编译选项，该选项可将编译生成的可执行程序命名为 outfile，如果不写该编译选项，默认情况下生成的可执行程序名为 a.exe（linux 下实际为 a.out 或无后缀名）。
3. 调用词法分析器 yylex()的 main 函数可以写在 lex 源程序的辅助函数部分，也可以写在其他的 c 文件中。如果 main 函数写在 main.c 中，则编译时需要和 lex.yy.c 一起编译链接，即编译链接命令为：gcc [-o outfile] lex.yy.c main.c –lfl
4. 运行可执行文件 a.exe 的命令（假设 a.exe 处于当前目录下，且忽略运行参数的情况）：./a.exe

   其中，./表示当前目录。如果 a.exe 处于其他路径，则运行时请给出完整路径名。（注意运行词法分析器的可执行文件时，如果可执行文件名不是 a.exe，而是其他名字，则主文件名中最好不要出现点（.），例如可执行文件名如果是 lex.yy.exe，则主文件名中含有点，容易出现问题，最好改名，例如改成 lexyy.exe。）

由于时间有限，该手册难免疏漏之处，如发现问题，请联系 gelin@ouc.edu.cn 指出。