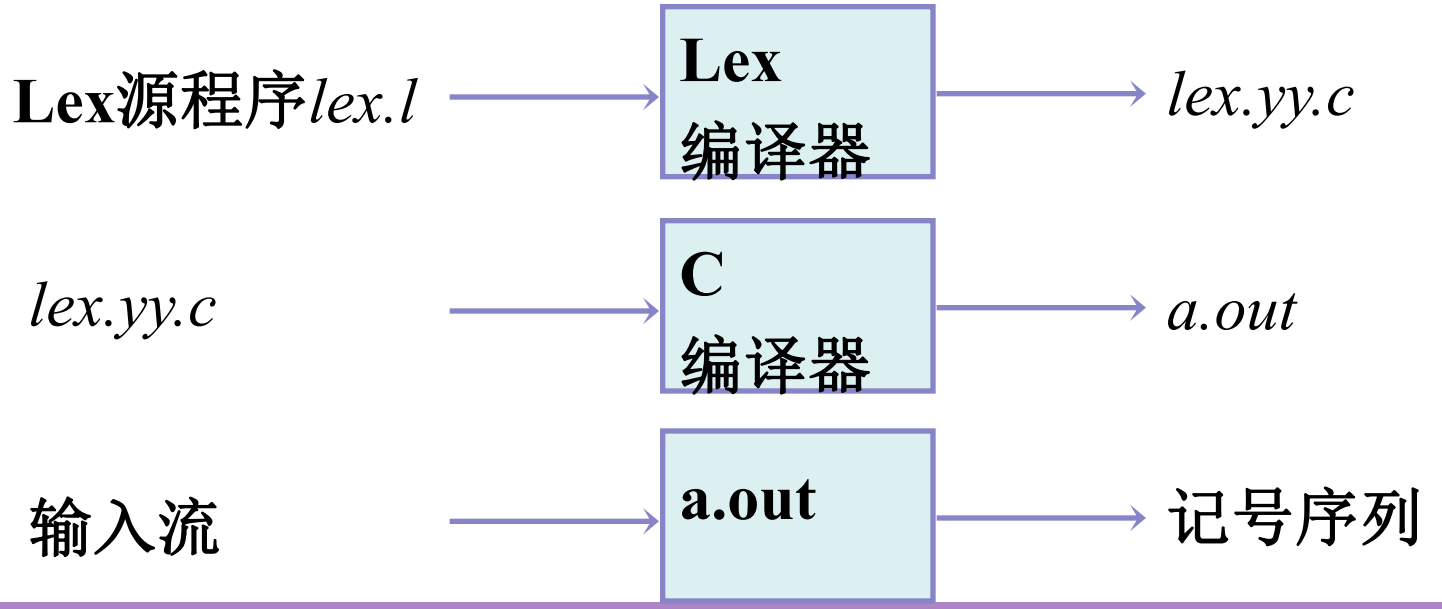
词法分析器的自动产生

用Lex建立词法分析器的步骤

Lex 编程可以分为三步：

- 以 Lex 可以理解的格式指定模式相关的动作。
- 在这一文件上运行lex文件，生成扫描器的 C 代码。
- 编译和链接 C 代码，生成可执行的扫描器。



Lex程序结构

一个 Lex 程序分为三个部分：

- 第一部分是 C 和 Lex 的全局声明
- 第二部分包括模式（C 代码）
- 第三部分是补充的 C 函数。 一般都有 main() 函数

这三个部分以%%来分界。

declarations

声明

%%

translation rules

转换规则

%%

auxiliary procedures

辅助过程

declarations 声明

可以提供 C 变量声明、常量定义、头文件包含等

```
%{
int wordCount = 0;
%}
```

标识(token)的正则表达式定义

```
chars           [A-Za-z]
words           {chars}+
```

translation rules 转换规则

Lex程序的转换规则是如下形式的语句：

p_1 $\{action_1\}$

p_2 $\{action_2\}$

… …

p_n $\{action_n\}$

其中，p_i是正则表达式，每个动作$action_i$表示匹配该表达式成功后，词法分析器要执行的程序段(用C语言编写) 应该执行的代码。这些代码都将放在函数**yylex()**中。

```
%%
{words}        { wordCount++; /* increase the word count by one*/ }
```

auxiliary procedures 辅助过程

```
%%
void main()
{
   yylex();        /* start the analysis*/
   printf("No of words:%d\n", wordCount);
}
```

Lex词法分析器工作方式

- 词法分析器，从尚未扫描的输入字符串中读字符，每次读入一个字符，直到发现能与某个正规表达式p_i匹配的最长前缀。
- 词法分析器执行action。
- 词法分析器这种不断查找词素，直到以显示的return调用结束工作的方式，使其可以方便的处理空白符和注释。
- 词法分析器只返回记号给语法分析器，带有与词素相关信息的属性值是通过全局变量yylval传递的。

Lex举例

正则表达式	记号	属性值
ws	**-**	**-**
if	**if**	**-**
then	**then**	**-**
else	**else**	**-**
id	**id**	**指向符号表表项的指针**
num	**num**	**指向符号表表项的指针**
<	**relop**	**LT**
<=	**relop**	**LE**
=	**relop**	**EQ**
<>	**relop**	**NE**
>	**relop**	**GT**
>=	**relop**	**GE**

Lex举例

```
%{
        /*符号常数定义
LT, LE, EQ, NE, GT, GE,
IF, THEN, ELSE, ID, NUMBER, RELOP */
%}
 /*正规定义*/
delim                   [ \t \n]
ws                              {delim}+
letter                  [A-Za-z]
digit                           [0-9]
id                              {letter}({letter}|{digit})*
number                  {digit}+(\.{digit}+)?(E[+\-]?{digit}+)?
%%
```

Lex举例

```
%%
{ws}                    {/*没有动作和返回值*/ }
if                              {return(IF);}
then                            {return(THEN);}
else                            {return(ELSE);}
{id}                            {yylval = install_id(); return(ID);}
{number}        {yylval = install_num(); return(NUMBER);}
“<”                             {yylval = LT; return(RELOP);}
“<=”                    {yylval = LE; return(RELOP);}
“=”                             {yylval = EQ; return(RELOP);}
“<>”                    {yylval = NE; return(RELOP);}
“>”                             {yylval = GT; return(RELOP);}
“>=”                    {yylval = GE; return(RELOP);}
%%
```

Lex举例

```
%%
install_id()
{
/*往符号表填入词素的过程，yytext指向词素的第一个字符，yyleng表示词素的长度。将词
 素填入符号表，返回指向该词素所在表项的指针*/
}
install_num()
{
/*与填写词素的过程类似，只不过词素是一个数。*/
}
```

Lex举例

```
%{
int wordCount = 0;
%}

chars               [A-za-z]
delim               [ \n\t]
whitespace          {delim}+
words               {chars}+
%%
{words}             { wordCount++; /* increase the word count by one*/ }

{whitespace}        { /* do nothing*/ }
\n|.                { /* gobble up */ }

%%
void main()
{
  yylex();          /* start the analysis*/
  printf("No of words:%d\n", wordCount);
}
```

C和Lex的全局声明

为字数统计程序声明一个整型变量，保存统计得到的字数

模式匹配规则

使用 C 语句来定义标记

匹配后的动作

Lex变量和函数

Lex 有几个函数和变量提供了不同的信息，可以用来编译实现复杂函数的程序。下面列出了一些变量和函数，以及它们的使用。

Lex 变量

yyin　　FILE* 类型。 指向 lexer 正在解析的当前文件。

yyout　　FILE* 类型。 指向记录 lexer 输出的位置。
缺省情况下，yyin 和 yyout 都指向标准输入和输出。

yytext　匹配模式的文本存储在这一变量中（char*）。

yyleng　给出匹配模式的长度。

Lex的匹配策略

elsen = 0;

1	else	n	=	0
2	elsen	=	0	

当输入的多个前缀与一个或多个模式匹配时，lex利用如下规则选择正确的词素：

总是选择最长的前缀

如果最长的可能前缀与多个模式匹配，总是选择在Lex程序中先被列出的模式。