# Lecture 3: Lexical Analysis ( Part II )

Xiaoyuan Xie 谢晓园
xxie@whu.edu.cn
计算机学院E301
课程主页：http://xiaoyuanxie.github.io/2017FallCS-Compilers/2017FallCSCompiler.html

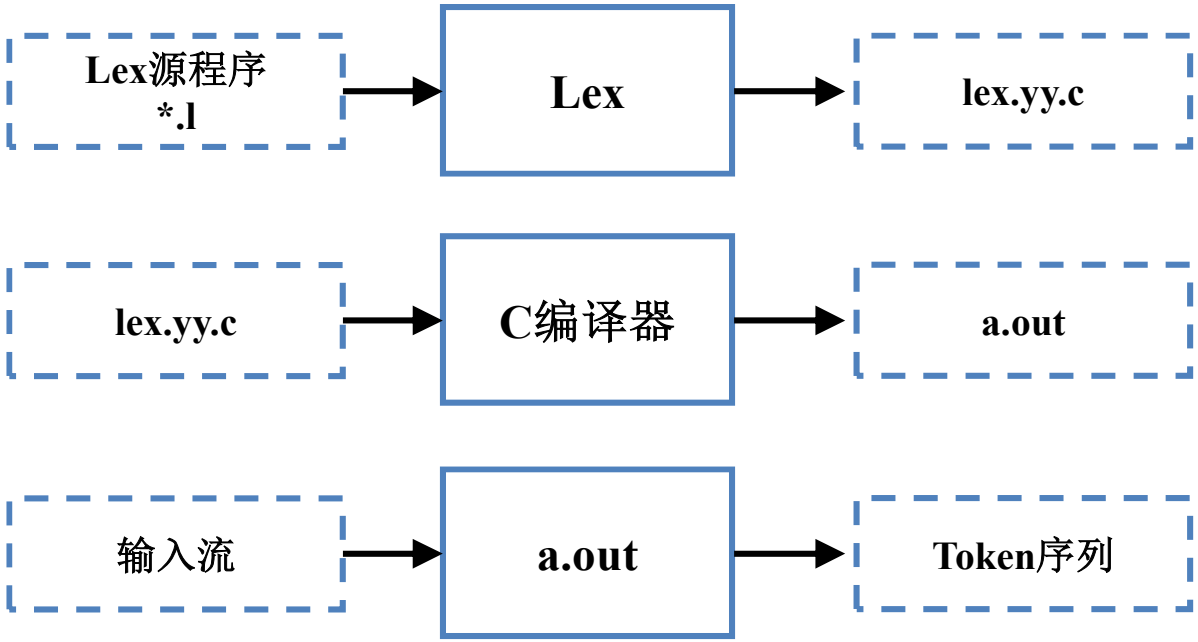

# 3.8 词法分析器的开发

# 3.8 词法分析器的开发

- LEX（Lexical Analyzer Generator）即词法分析器的生成器，是由贝尔实验室于1972年在UNIX操作系统上首次开发的。最新版本是FLEX(Fast Lexical Analyzer Genrator)

- 工作原理：LEX通过对Lex源文件(.l文件)的扫描，经过宏替换将规则部分的正则表达式转换成与之等价的DFA，并产生DFA的状态转换矩阵；利用该矩阵和Lex源文件中的C代码一起产生一个名为yylex()的词法分析函数，并将yylex()函数拷贝到输出文件lex.yy.c中。

# 3.8 词法分析器的开发




用Lex创建一个词法分析器的过程

# 3.8 词法分析器的开发

- Lex源文件

*声明部分*

*%%*

*转换规则*

*%%*

*辅助函数*

**动作中需要使用的函数**

```
int  Change()
{  /*将字符串形式的常数
     转换成整数形式*/
}
```

*%{ 常量*

*}%*

*正则定义*

模式 {动作}：

- 模式是一个正则表达式或者正则定义
- 动作通常是C语言代码，表示匹配该表达式后应该执行的代码。

**yylval：token的值**
**yytext：token的lexeme**
**yyleng：lexeme的长度**

```
%{ ID,NUM,IF,ADD
}%
letter  [A-Za-z]
digit  [0-9]
id     {letter}({letter}|{digit})*
num {digit}+
if   {return (IF);}
+   {return(ADD);}
{id} {yylval = strcpy(yytext,
       yylength); return(ID);  }
{num} {yylval = Change();
       return(NUM);}
```

# 3.8 词法分析器的开发

- Lex例子

| 词素 | 词法单元名字 | 属性值 |
|---|---|---|
| Any *ws* | – | – |
| if | if | – |
| then | then | – |
| else | else | – |
| Any *id* | id | 指向符号表条目的指针 |
| Any *number* | number | 指向符号表条目的指针 |
| < | relop | LT |
| <= | relop | LE |
| = | relop | EQ |
| <> | relop | NE |
| > | relop | GT |
| >= | relop | GE |

图 3-12　词法单元、它们的模式以及属性值

```
%{
    /* definitions of manifest constants
    LT, LE, EQ, NE, GT, GE,
    IF, THEN, ELSE, ID, NUMBER, RELOP */
%}

/* regular definitions */
delim     [ \t\n]
ws        {delim}+
letter    [A-Za-z]
digit     [0-9]
id        {letter}({letter}|{digit})*
number    {digit}+(\.{digit}+)?(E[+-]?{digit}+)?

%%

{ws}      {/* no action and no return */}
if        {return(IF);}
then      {return(THEN);}
else      {return(ELSE);}
{id}      {yylval = (int) installID(); return(ID);}
{number}  {yylval = (int) installNum(); 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);}

%%

int installID() {/* function to install the lexeme, whose
                    first character is pointed to by yytext,
                    and whose length is yyleng, into the
                    symbol table and return a pointer
                    thereto */
}

int installNum() {/* similar to installID, but puts numer-
                     ical constants into a separate table */
}
```

图 3-23　识别图 3-12 中的词法单元的 Lex 程序

# 3.8 词法分析器的开发

## ■ 冲突解决

当输入与长度不同的多个模式匹配时，Lex选择长模式进行匹配

当输入与长度相同的多个模式匹配时，Lex选择列于前面的模式进行匹配

```
%%
program     printf("%s\n",yytext);/*模式1*/
procedure    printf("%s\n",yytext);/*模式2*/
[a-z][a-z0-9]*  printf("%s\n",yytext);/*模式3*/
```

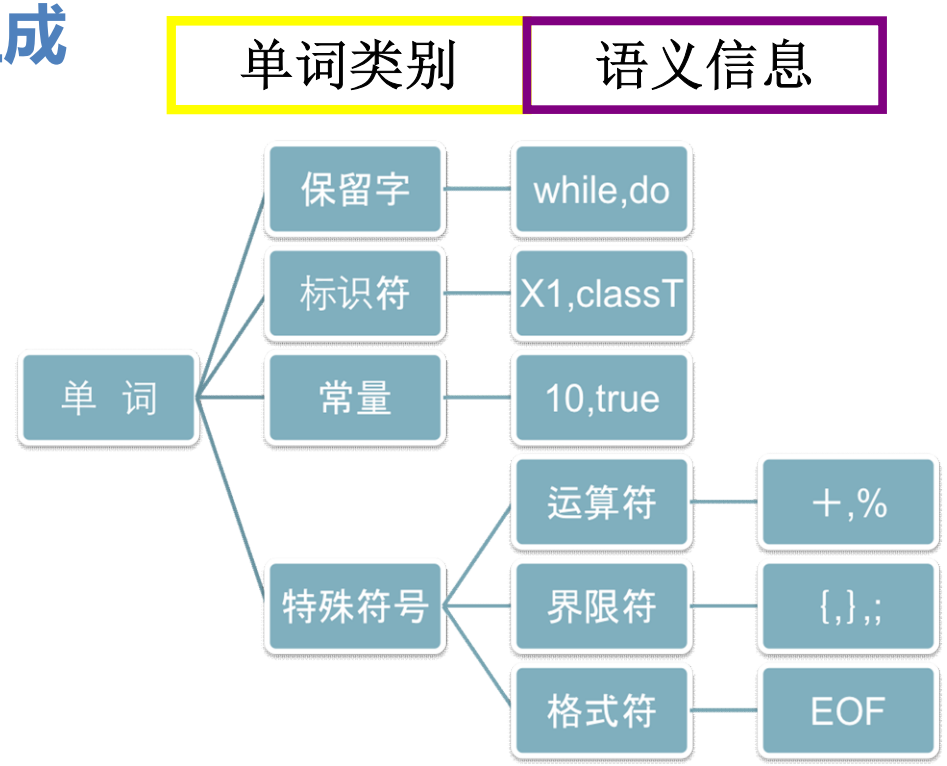
当输入串为“programming”时，模式1（匹配“program”）和模式3（“programming”）都匹配，但会选择匹配串长的模式3。

当输入串为“program”时，因为模式1和模式3匹配的串长度相等故会选择模式1.

# 3.8 词法分析器的开发

## 需要定义的词法组成

## TOKEN结构图



# Thank you!