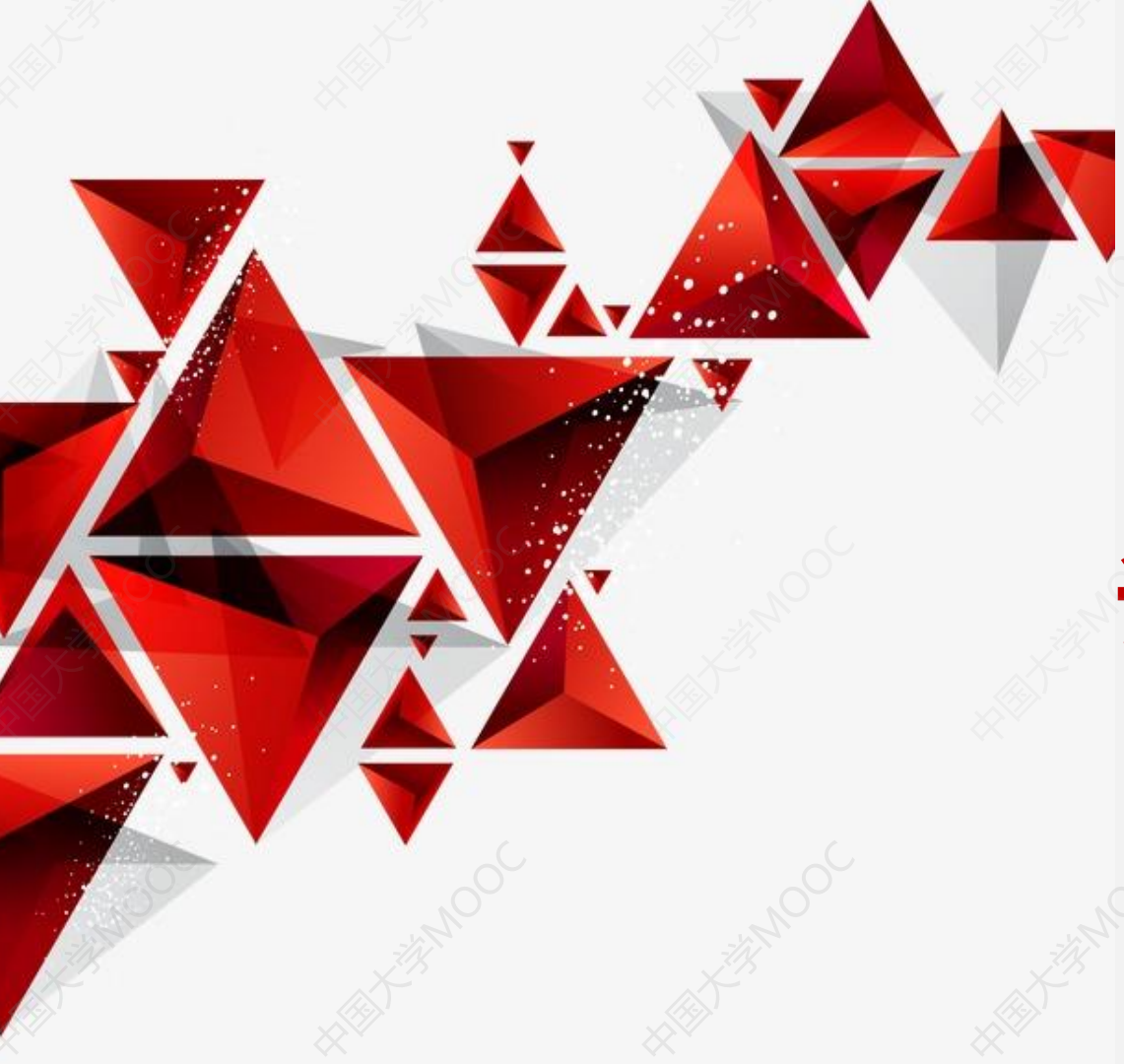
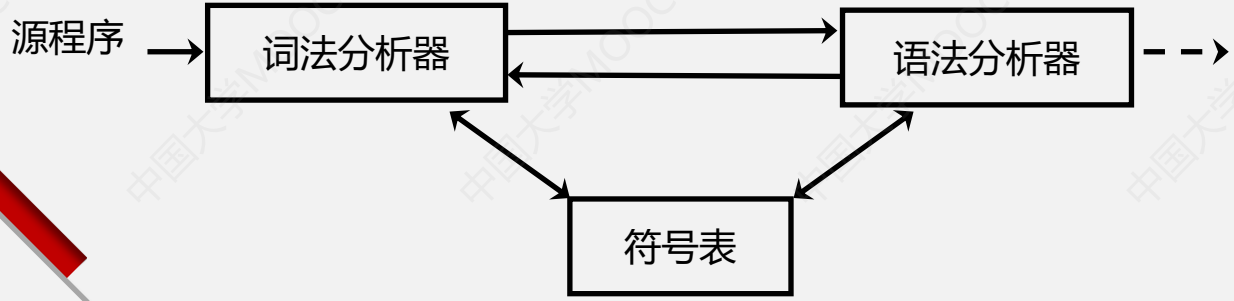


编译技术

词法分析

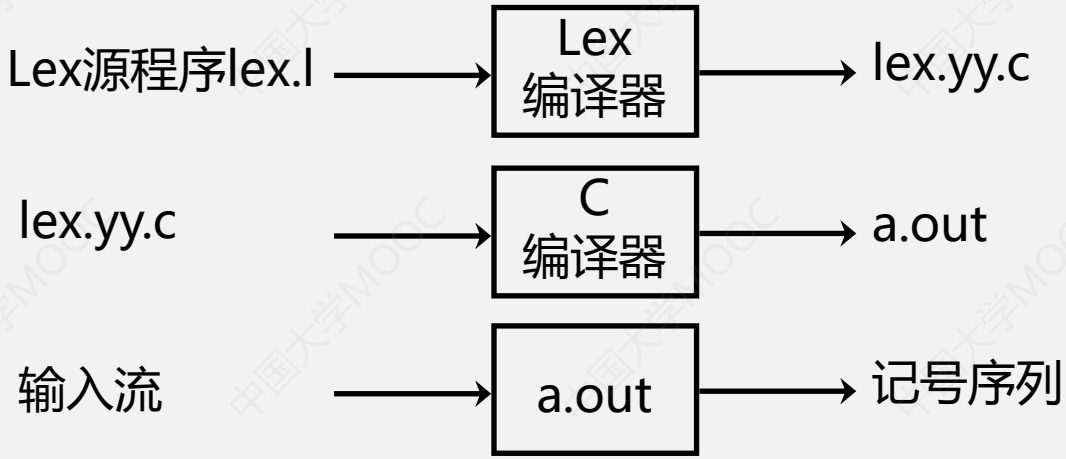
大连理工大学软件学院



本讲纲要

- 词法分析器的生成器Lex

用Lex建立词法分析器的步骤



词法分析器的生成器

Lex的工作原理

- 根据描述匹配模式的正则表达式构造出DFA，而后基于此DFA生成词法分析程序的主控制结构

词法分析器的生成器

- Lex的工作原理
  - 根据描述匹配模式的正则表达式构造出DFA，而后基于此DFA生成词法分析程序的主控制结构
- Lex的实现
  - 宿主语言：C, C++等
  - 每个匹配动作相关的代码被放在对应的状态的处理代码块中

词法分析器的生成器

- Lex的工作原理
  - 根据描述匹配模式的正则表达式构造出DFA，而后基于此DFA生成词法分析程序的主控制结构
- Lex的实现
  - 宿主语言：C, C++等
  - 每个匹配动作相关的代码被放在对应的状态的处理代码块中
- Lex工具
  - lex, flex
  - JLex
  - TP Lex
  - …

词法分析器的生成器

- Lex程序包括三个部分
  - 声明
  - %%
  - 翻译规则
  - %%
  - 辅助过程

词法分析器的生成器

- Lex程序包括三个部分
  - 声明
  - %%
  - 翻译规则
  - %%
  - 辅助过程

词法分析器的生成器

- Lex程序包括三个部分
  - 声明
  - %%
  - 翻译规则
  - %%
  - 辅助过程

词法分析器的生成器

- Lex程序包括三个部分
  - 声明
  - %%
  - 翻译规则
  - %%
  - 辅助过程
- Lex程序的翻译规则

p_1	{动作1}
p_2	{动作2}
…	…
p_n	{动作n}

词法分析器的生成器

- Lex程序包括三个部分
  - 声明
  - %%
  - 翻译规则
  - %%
  - 辅助过程
- Lex程序的翻译规则

p_1	{动作1}
p_2	{动作2}
…	…
p_n	{动作n}

例---声明部分

```
%{
/* 常量LT, LE, EQ, NE, GT, GE, WHILE, DO, ID, NUMBER,
    RELOP的定义*/
%}
/* 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}+)?
```

例---声明部分

```
%{
/*  常量LT,  LE,  EQ,  NE,  GT,  GE,  WHILE,  DO,  ID,  NUMBER,
    RELOP的定义*/
%}
/* 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}+)?
```

例---声明部分

```
%{
/*  常量LT,  LE,  EQ,  NE,  GT,  GE,  WHILE,  DO,  ID,  NUMBER,
    RELOP的定义*/
%}
/* 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}+)?
```

例---声明部分

```
%{
/* 常量LT, LE, EQ, NE, GT, GE, WHILE, DO, ID, NUMBER,
    RELOP的定义*/
%}
/* 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}+)?
```

例---翻译规则部分

{ws}	{/* 没有动作，也不返回 */}
while	{return (WHILE);}
do	{return (DO);}
{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);}

例---翻译规则部分

{ws}	{/* 没有动作，也不返回 */}
while	{return (WHILE);}
do	{return (DO);}
{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);}

例---翻译规则部分

{ws}	{/* 没有动作，也不返回 */}
while	{return (WHILE);}
do	{return (DO);}
{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);}

例---翻译规则部分

{ws}	{/* 没有动作，也不返回 */}
while	{return (WHILE);}
do	{return (DO);}
{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);}

例---翻译规则部分

{ws}	{/* 没有动作，也不返回 */}
while	{return (WHILE);}
do	{return (DO);}
{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);}

例---辅助过程部分

```
install_ id ( ) {
        /*  把词法单元装入符号表并返回指针。
        yytext指向该词法单元的第一个字符，
        yyleng给出的它的长度                    */
}
```

例---辅助过程部分

```
install_ id ( ) {
        /*  把词法单元装入符号表并返回指针。
        yytext指向该词法单元的第一个字符，
        yyleng给出的它的长度                    */
}

install_num ( ) {
        /* 类似上面的过程，但词法单元不是标识符而是数 */
}
```

用 Lex 定义常规表达式

.	匹配任意字符，除了\n
-	指范围[A-Za-z0-9]
$	行的结尾
{ }	模式可能出现的次数，例如A{1,3}表示可能出现1次或3次
^	否定,操作符^只能出现在左中括号后的第一个字符位置处[^abc]
* | ?+等	常用的闭包，逻辑或等操作

用 Lex 定义常规表达式

.	匹配任意字符，除了\n
-	指范围[A-Za-z0-9]
$	行的结尾
{ }	模式可能出现的次数，例如A{1,3}表示可能出现1次或3次
^	否定,操作符^只能出现在左中括号后的第一个字符位置处[^abc]
* | ?+等	常用的闭包，逻辑或等操作

Lex中重要的外部变量

- yytext：外部字符数组，其内容是当前被某个规则匹配的字符串
- yyleng：当前yytext中的字符的个数

Lex中重要的外部变量

- yytext：外部字符数组，其内容是当前被某个规则匹配的字符串
- yyleng：当前yytext中的字符的个数

- 例：

```
[a-zA-Z]+     printf(“word=%s, length=%d”, yytext, yyleng);
```

Lex中重要的外部变量

- yytext：外部字符数组，其内容是当前被某个规则匹配的字符串
- yyleng：当前yytext中的字符的个数

- 例：

[a-zA-Z]+　　printf（“word=%s, length=%d”, yytext, yyleng);

[a-zA-Z]+　　printf（“%s”, yytext);
可简写[a-zA-Z]+　ECHO;

Lex中重要的变量

- yyin
  - 类型: FILE *
  - 词法分析的输入文件
- yyout
  - 类型: FILE *
  - 词法分析的输出文件

- 以上两个经常和函数yywrap()连用，如果函数的返回值是1，就停止解析。
- 因此它可以用来解析多个文件。
- 代码可以写在第三段，这就能够解析多个文件。
- 方法是使用 yyin 文件指针指向不同的文件，直到所有的文件都被解析。
- 最后，yywrap() 可以返回 1 来表示解析的结束。

Lex中重要的变量

- yyin
  - 类型: FILE *
  - 词法分析的输入文件
- yyout
  - 类型: FILE *
  - 词法分析的输出文件
- yylineno
  - 给出当前的行数信息

- 以上两个经常和函数yywrap()连用，如果函数的返回值是1，就停止解析。
- 因此它可以用来解析多个文件。
- 代码可以写在第三段，这就能够解析多个文件。
- 方法是使用 yyin 文件指针指向不同的文件，直到所有的文件都被解析。
- 最后，yywrap() 可以返回 1 来表示解析的结束。

Lex中识别规则二义性处理

- 能匹配最多字符的规则优先

Lex中识别规则二义性处理

- 能匹配最多字符的规则优先

integer　keyword action ...;

[a−z]+　identifier action ...;

当输入为integers时，匹配[a−z]+

Lex中识别规则二义性处理

- 能匹配最多字符的规则优先
- 能匹配相同数目的字符的规则，书写顺序在前的优先

Lex中识别规则二义性处理

- 能匹配最多字符的规则优先
- 能匹配相同数目的字符的规则，书写顺序在前的优先

假设需要计算输入文本中she和he的个数

```
she     {s++; REJECT;}
he      {h++; REJECT;}
\n      |
.       ;
```

. 匹配任意字符，除了 \n

简单的例子

删除输入中每行结尾处所有空白符

```
%%
[ \t]+$        ;
```

简单的例子

- 删除输入中每行结尾处所有空白符

```
%%
[ \t]+$        ;
```

- 如果要将字符串中的空格或者制表符转换为单个空格，需要增加一条规则：

```
%%
[ \t]+$        ;
[ \t]+         printf( “ ” );
```

上机实验例子

example.l

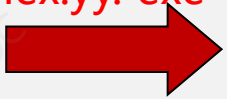
```
%{
int num_lines = 0, num_chars = 0;
%}
%%
\n   ++num_lines; ++num_chars;
.    ++num_chars;

%%
main()
{
        yylex();
        printf("# of lines = %d, # of chars = %d\n ",num_lines,num_chars);
}
```

上机实验例子

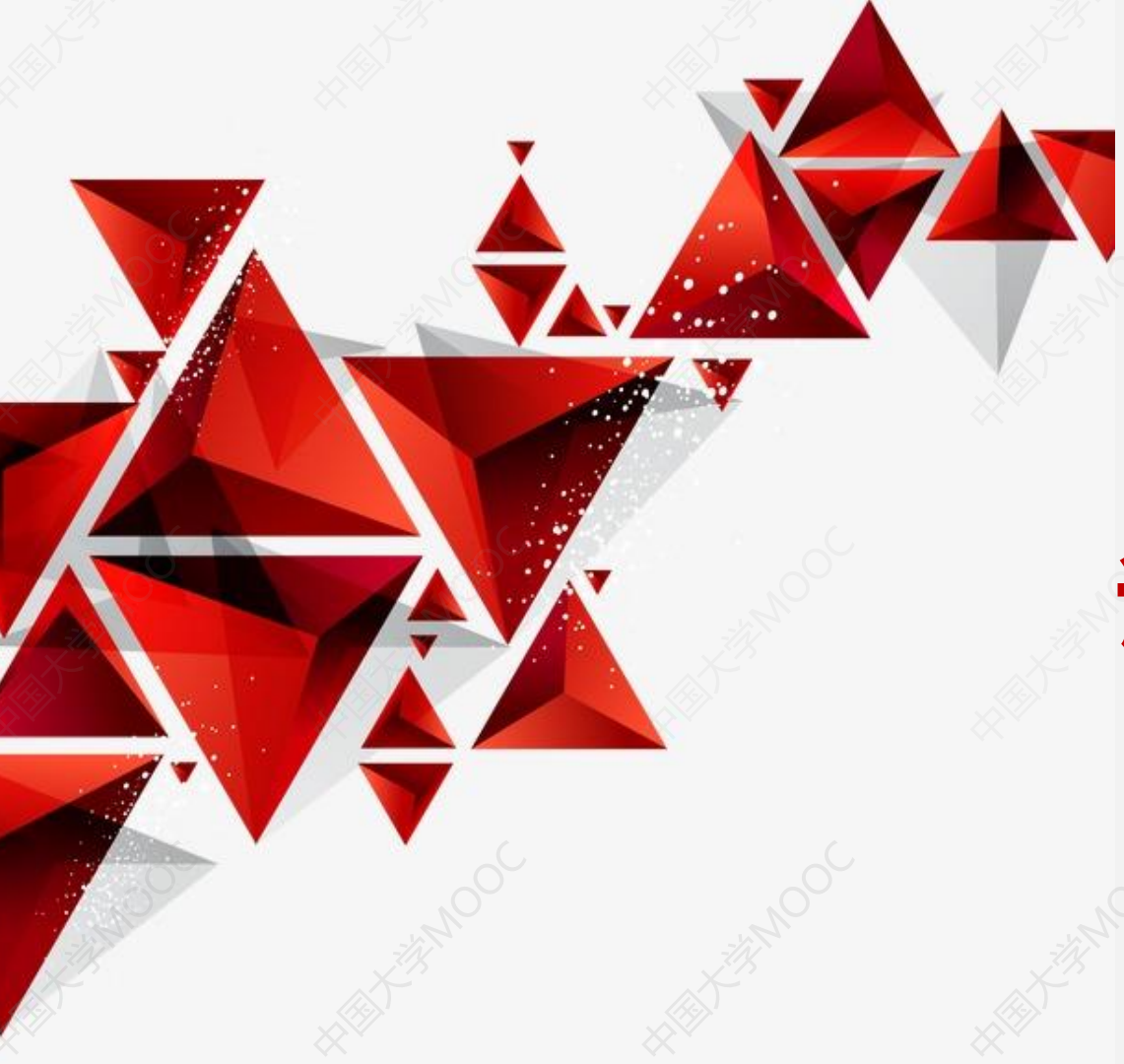
example.l

hello world
wo ai tian an men
hello world i love

lex.yy. exe



# of lines = 3, # of chars = 49



谢 谢 ！

大连理工大学软件学院