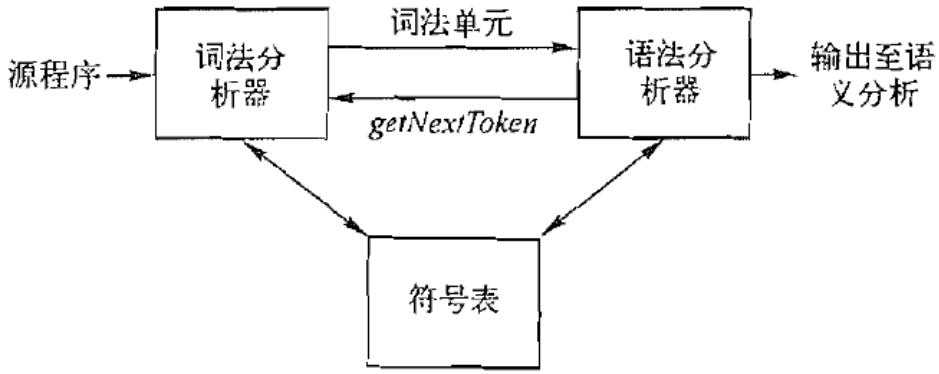
词法分析

魏恒峰

hfwei@nju.edu.cn

2020 年 11 月 4 日

输入: 程序文本/字符串 s & **词法单元 (token) 的规约**



输出: 词法单元流

token : ⟨token-class, attribute-value⟩

词法单元	非正式描述	词素示例
if	字符 `i`, `f`	`if`
else	字符 `e`, `l`, `s`, `e`	`else`
comparison	`<` 或 `>` 或 `<=` 或 `>=` 或 `==` 或 `!=`	`<=`, `!=`
id	字母开头的字母 / 数字串	`pi`, `score`, `D2`
number	任何数字常量	`3.14159`, `0`, `6.02e23`
literal	在两个 `"`之间，除`"` 以外的任何字符	`"core dumped"`

token : ⟨token-class, attribute-value⟩

词法单元	非正式描述	词素示例
if	字符 i, f	if
else	字符 e, l, s, e	else
comparison	< 或 > 或 <= 或 >= 或 == 或 !=	<=, !=
id	字母开头的字母 / 数字串	pi, score, D2
number	任何数字常量	3.14159, 0, 6.02e23
literal	在两个 "之间，除" 以外的任何字符	"core dumped"

int/if 关键词

ws 空格、制表符、换行符

comment “//” 开头的一行注释或者 “/* */” 包围的多行注释

```c
int main(void)
{
    printf("hello, world\n");
}
```

```c
int main(void)
{
    printf("hello, world\n");
}
```

int ws **main/id** **LP** **void** **RP** ws

LB ws

ws **id** **LP** **literal** **RP** **SC**ws

RB

```c
int main(void)
{
    printf("hello, world\n");
}
```

int ws **main/id** **LP** **void** **RP** ws

LB ws

ws **id** **LP** **literal** **RP** **SC**ws

RB

本质上, 这就是一个**字符串（匹配/识别）算法**

词法分析器的三种设计方法

手写词法分析器

词法分析器的生成器

自动化词法分析器

生产环境下的编译器 (如 gcc) 通常选择**手写词法分析器**

识别字符串 s 中符合**某种词法单元模式**的**所有词素**

```
if ab42>=42
    xyz =3.14
else xyz = 2.718
```

ws **if** **else** **id** **integer** **real** **relop**

识别字符串 s 中符合**某种词法单元模式**的**所有词素**

```
if ab42>=42
    xyz =3.14
else xyz = 2.718
```

ws　**if**　**else**　**id**　**integer**　**real**　**relop**

识别字符串 s 中符合**某种词法单元模式**的**第一个词素**

识别字符串 s 中符合**某种词法单元模式**的**所有词素**

```
if ab42>=42
    xyz =3.14
else xyz = 2.718
```

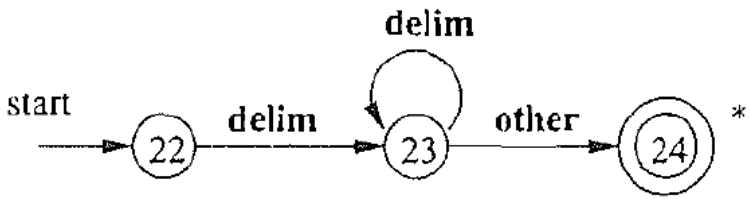
ws　**if**　**else**　**id**　**integer**　**real**　**relop**

识别字符串 s 中符合**某种词法单元模式**的**第一个词素**

识别字符串 s 中符合**特定词法单元模式**的**第一个词素**

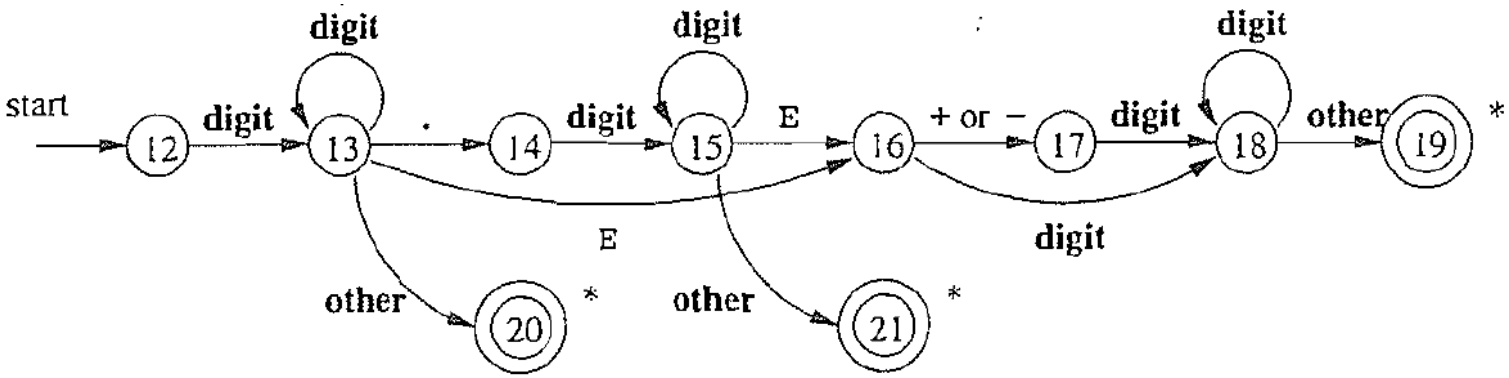
识别字符串 s 中符合**特定词法单元模式**的**第一个词素**

ws: **blank** **tab** **newline**



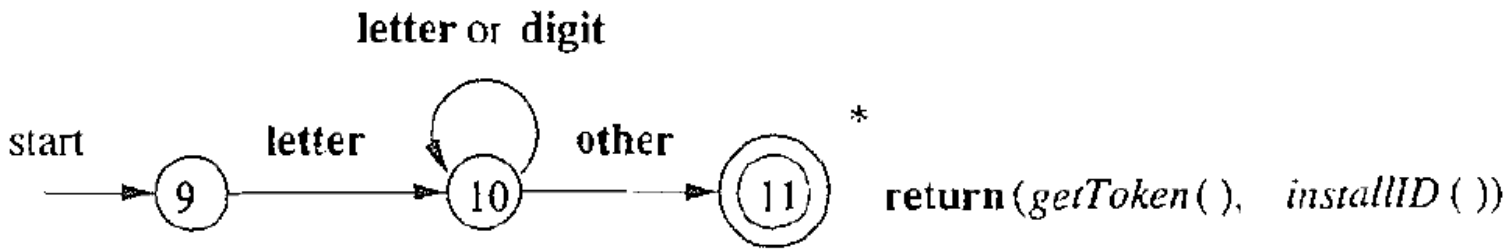
识别字符串 s 中符合**特定词法单元模式**的**第一个词素**

num: 整数 (允许以 0 开头)



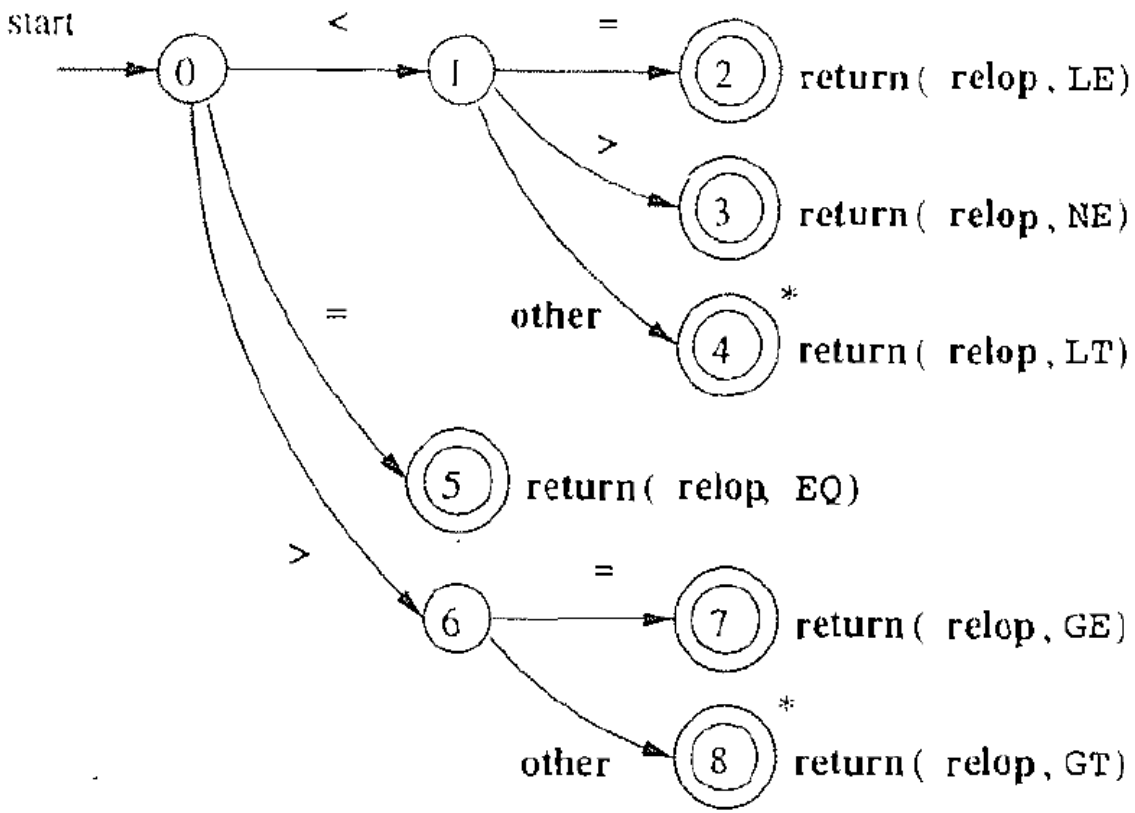
识别字符串 s 中符合**特定词法单元模式**的**第一个词素**

id: 字母开头的字母/数字串

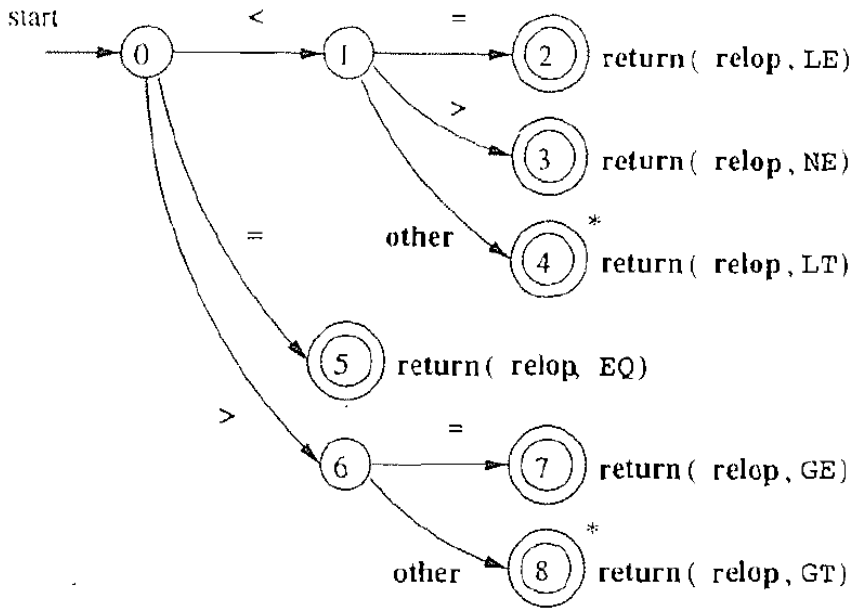


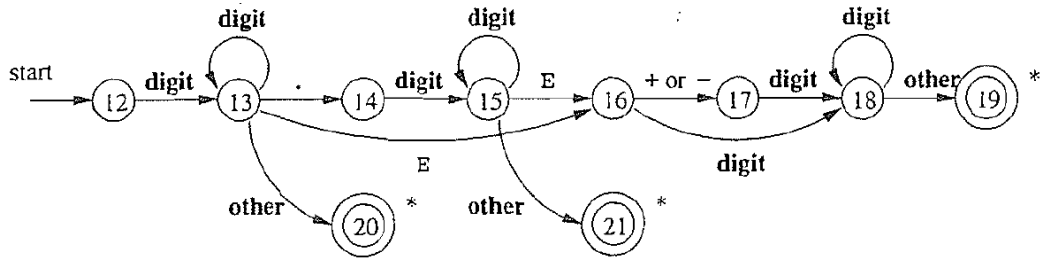
识别字符串 s 中符合**特定词法单元模式**的**第一个词素**

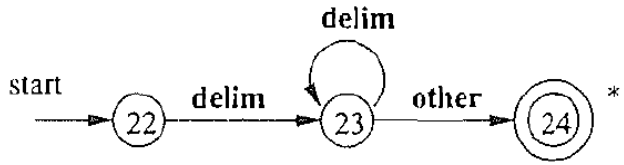
relop: $<$ $>$ $<=$ $>=$ $==$ $<>$

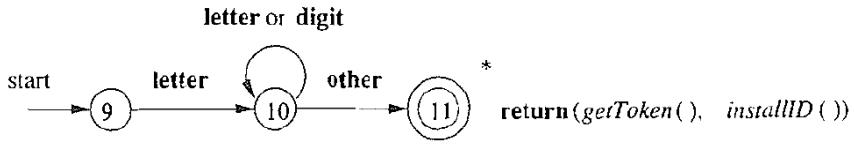


识别字符串 s 中符合**某种词法单元模式**的**第一个词素** (SCAN())

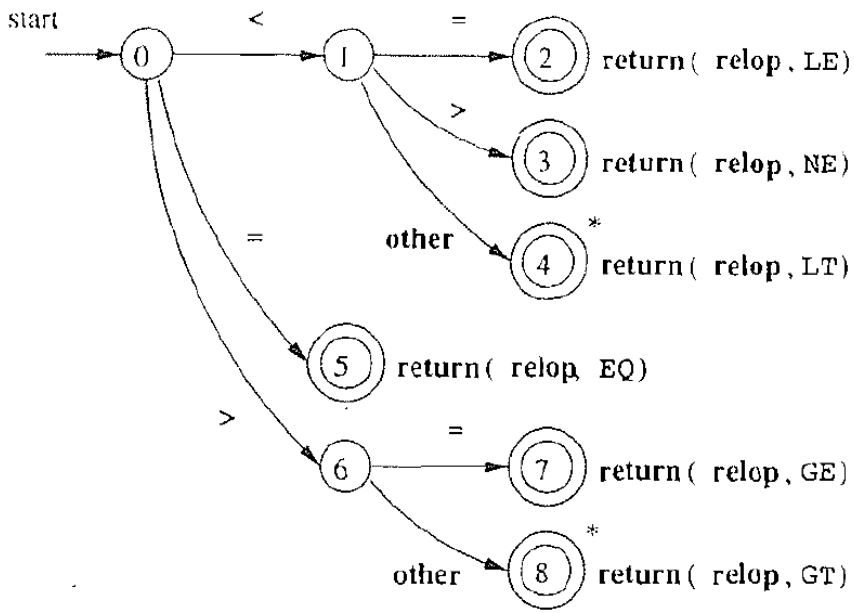


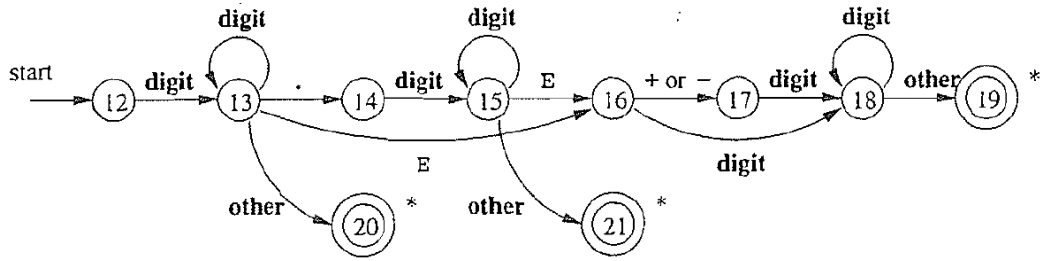


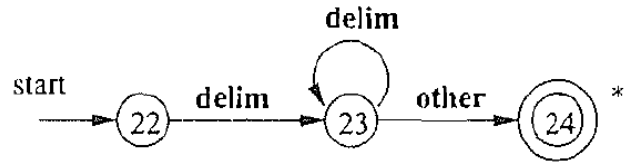


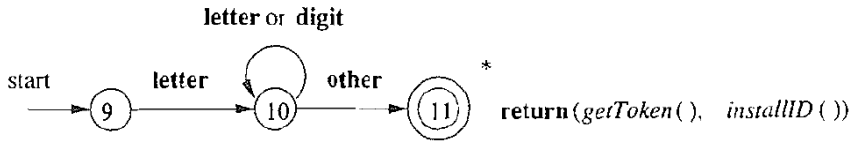


识别字符串 s 中符合**某种词法单元模式**的**第一个词素** (SCAN())



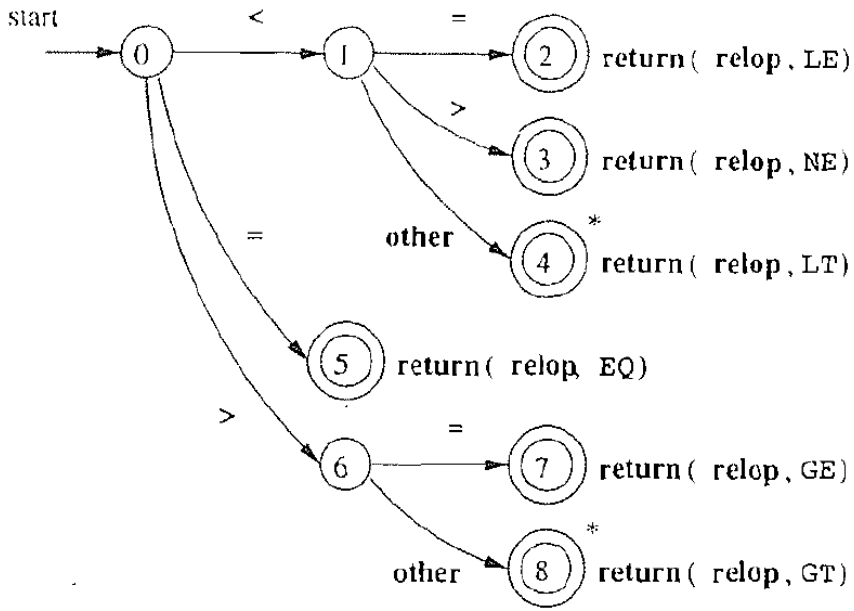


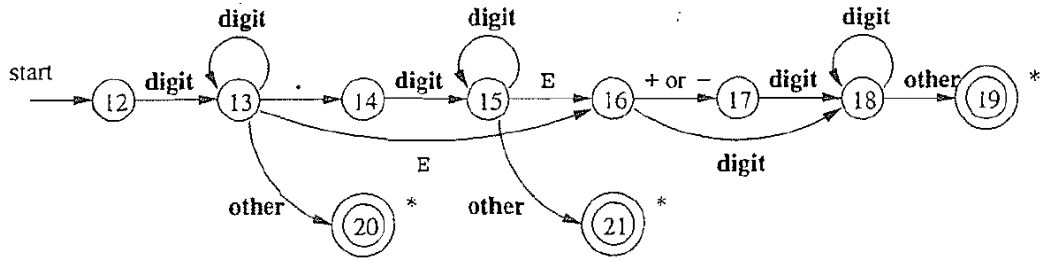


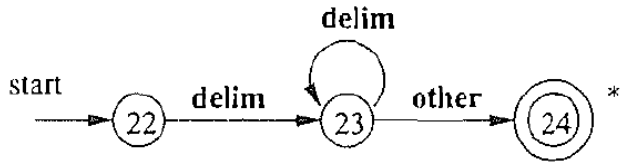


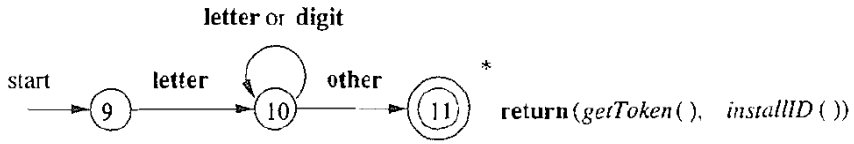
根据**下一个字符**即可判定词法单元的类型

识别字符串 s 中符合**某种词法单元模式**的**第一个词素** (SCAN())









根据**下一个字符**即可判定词法单元的类型

否则，报告**该字符有误**，并忽略该字符

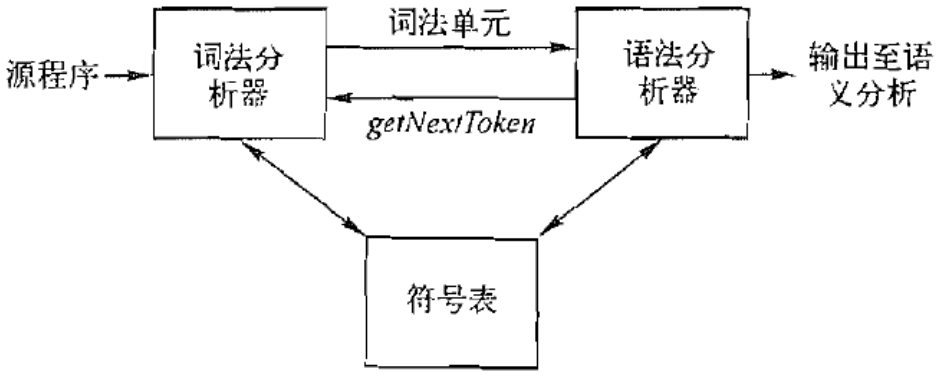
识别字符串 s 中符合**某种词法单元模式**的**所有词素**

最外层循环调用 SCAN()

或者, 由语法分析器按需调用 SCAN()

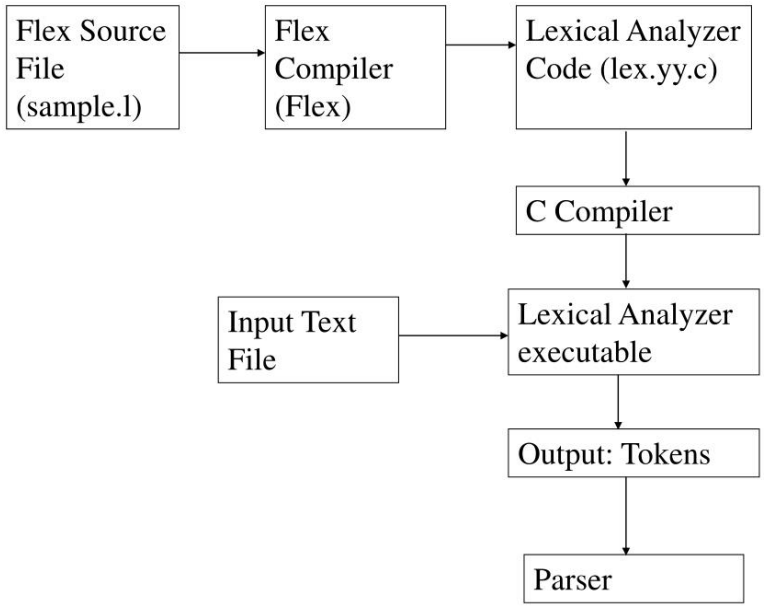

flex

输入: 程序文本/字符串 s & 词法单元的规约



输出: 词法单元流

输入：词法单元的规约

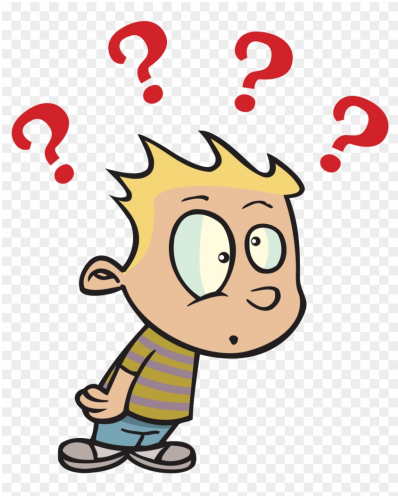


输出：词法分析器

词法单元的规约

词法单元	非正式描述	词素示例
if	字符 i, f	if
else	字符 e, l, s, e	else
comparison	< 或 > 或 <= 或 >= 或 == 或 !=	<=, !=
id	字母开头的字母 / 数字串	pi, score, D2
number	任何数字常量	3.14159, 0, 6.02e23
literal	在两个 "之间，除" 以外的任何字符	"core dumped"

词法单元的规约



词法单元	非正式描述	词素示例
if	字符 i, f	if
else	字符 e, l, s, e	else
comparison	< 或 > 或 <= 或 >= 或 == 或 !=	<=, !=
id	字母开头的字母 / 数字串	pi, score, D2
number	任何数字常量	3.14159, 0, 6.02e23
literal	在两个 "之间，除" 以外的任何字符	"core dumped"

我们需要词法单元的**形式化**规约

id: 字母开头的字母/数字串

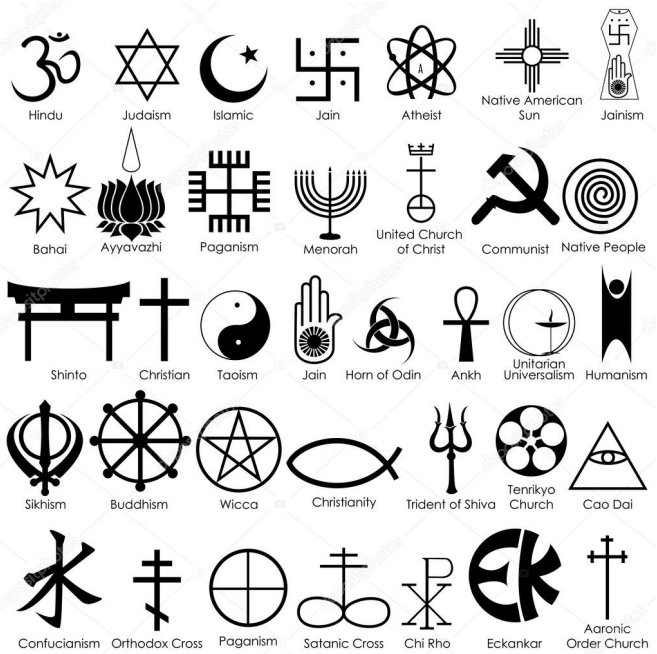
id 定义了一个集合, 我们称之为**语言 (Language)**

它使用了字母与数字等符号集合, 我们称之为**字母表 (Alphabet)**

该语言中的每个元素 (即, 标识符) 称为**串 (String)**

Definition (字母表)

字母表 Σ 是一个有限的符号集合。



Definition (串)

字母表 Σ 上的串 (s) 是由 Σ 中符号构成的一个**有穷**序列。

空串：$|\epsilon| = 0$

Definition (串上的 “连接” 运算)

$$x = dog, y = house \qquad xy = doghouse$$

$$s\epsilon = \epsilon s = s$$

Definition (串上的 “连接” 运算)

$$x = dog, y = house \qquad xy = doghouse$$

$$s\epsilon = \epsilon s = s$$

Definition (串上的 “指数” 运算)

$$s^0 \triangleq \epsilon$$

$$s^i \triangleq ss^{i-1}, i > 0$$

Definition (语言)

语言是给定字母表 Σ 上一个任意的**可数的**串集合。

$$\emptyset \qquad \{\epsilon\}$$

Definition (语言)

语言是给定字母表 Σ 上一个任意的**可数的**串集合。

$$\emptyset \qquad \{\epsilon\}$$

$$\mathbf{id} : \{a, b, c, a1, a2, \dots\}$$

$$\mathbf{ws} : \{\mathbf{blank}, \mathbf{tab}, \mathbf{newline}\}$$

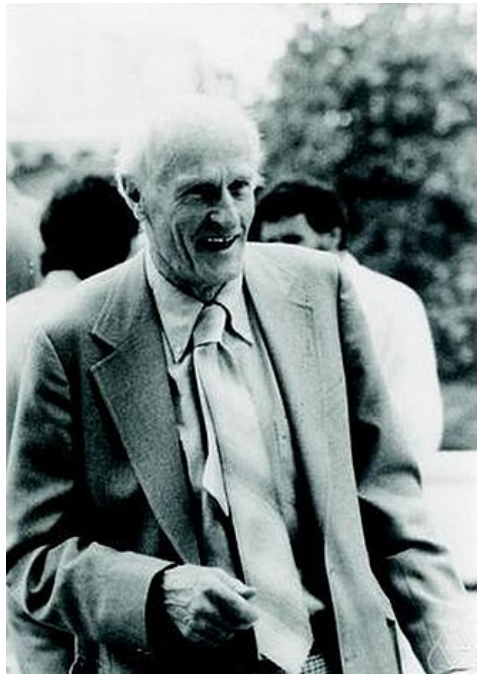
$$\mathbf{if} : \{if\}$$

语言是串的集合

因此, 我们可以通过集合操作**构造**新的语言。

运算	定义和表示
L和M 的并	$L \cup M \doteq \{s \mid s$ 属于L 或者s 属于 $M\}$
L和M 的连接	$LM = \{st \mid s$ 属于 L且　t 属于 $M\}$
L的 Kleene 闭包	$L^* = \cup_{i=0}^{\infty} L^i$
L的正闭包	$L^* = \cup_{i=1}^{\infty} L^i$

L^* 允许我们构造**无穷**集合



Stephen Kleene
(1909 ~ 1994)

$$L = \{A, B, \ldots, Z, a, b, \ldots, z\}$$

$$D = \{0, 1, \ldots, 9\}$$

运算	定义和表示
L和M 的并	$L \cup M \doteq \{s \mid s$ 属于L 或者s 属于 $M\}$
L和M 的连接	$LM = \{st \mid s$ 属于 L且　t 属于 $M\}$
L的 Kleene 闭包	$L^* = \cup_{i=0}^{\infty} L^i$
L的正闭包	$L^* = \cup_{i=1}^{\infty} L^i$

$$L = \{A, B, \ldots, Z, a, b, \ldots, z\}$$

$$D = \{0, 1, \ldots, 9\}$$

运算	定义和表示
L和M 的并	$L \cup M \doteq \{s \mid s$ 属于L 或者s 属于 $M\}$
L和M 的连接	$LM = \{st \mid s$ 属于 L且　t 属于 $M\}$
L的 Kleene 闭包	$L^* = \cup_{i=0}^{\infty} L^i$
L的正闭包	$L^* = \cup_{i=1}^{\infty} L^i$

$L \cup D$　　LD　　L^4　　L^*　　D^+

$L(L \cup D)^*$

$$\mathbf{id} : L(L \cup D)^*$$

如何更简洁地描述该 **id** 语言?

$$\textbf{id} : L(L \cup D)^*$$

如何更简洁地描述该 **id** 语言?

下面向大家隆重介绍简洁、优雅、强大的**正则表达式**

每个正则表达式 r 对应一个正则语言 $L(r)$

正则表达式是**语法**, 正则语言是**语义**

Definition (正则表达式)

给定字母表 Σ, Σ 上的正则表达式**由且仅由**以下规则定义:

(1) ϵ 是正则表达式;

(2) $\forall a \in \Sigma$, a 是正则表达式;

(3) 如果 r 是正则表达式, 则 (r) 是正则表达式;

(4) 如果 r 与 s 是正则表达式, 则 $r|s$, rs, r^* 也是正则表达式。

运算优先级: $() \succ * \succ$ 连接 $\succ |$

$$(a)|((b)^*(c)) \equiv a|b^*c$$

每个正则表达式 r 对应一个正则语言 $L(r)$

Definition (正则表达式对应的正则语言)

$$L(\epsilon) = \{\epsilon\} \tag{1}$$

$$L(a) = \{a\}, a \in \Sigma \tag{2}$$

$$L((r)) = L(r) \tag{3}$$

$$L(r|s) = L(r) \cup L(s) \quad L(rs) = L(r)L(s) \quad L(r^*) = (L(r))^* \tag{4}$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$L((a|b)(a|b))$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$L((a|b)(a|b))$$

$$L(a^*)$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$L((a|b)(a|b))$$

$$L(a^*)$$

$$L((a|b)^*)$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$L((a|b)(a|b))$$

$$L(a^*)$$

$$L((a|b)^*)$$

$$L(a|a^*b)$$


THAT'S IT?
TOO EASY.
memegenerator.net

表达式	匹配	例子
c	单个非运算符字符 c	a
\c	字符 c 的字面值	*
"s"	串 s 的字面值	"**"
.	除换行符以外的任何字符	a.*b
^	一行的开始	^abc
$	行的结尾	abc$
[s]	字符串 s 中的任何一个字符	[abc]
[^s]	不在串 s 中的任何一个字符	[^abc]
$r*$	和 r 匹配的零个或多个串连接成的串	a*
$r+$	和 r 匹配的一个或多个串连接成的串	a+
$r?$	零个或一个 r	a?
$r\{m,n\}$	最少 m 个，最多 n 个 r 的重复出现	a{1,5}
r_1r_2	r_1 后加上 r_2	ab
$r_1 \mid r_2$	r_1 或 r_2	a|b
(r)	与 r 相同	(a|b)
r_1/r_2	后面跟有 r_2 时的 r_1	abc/123

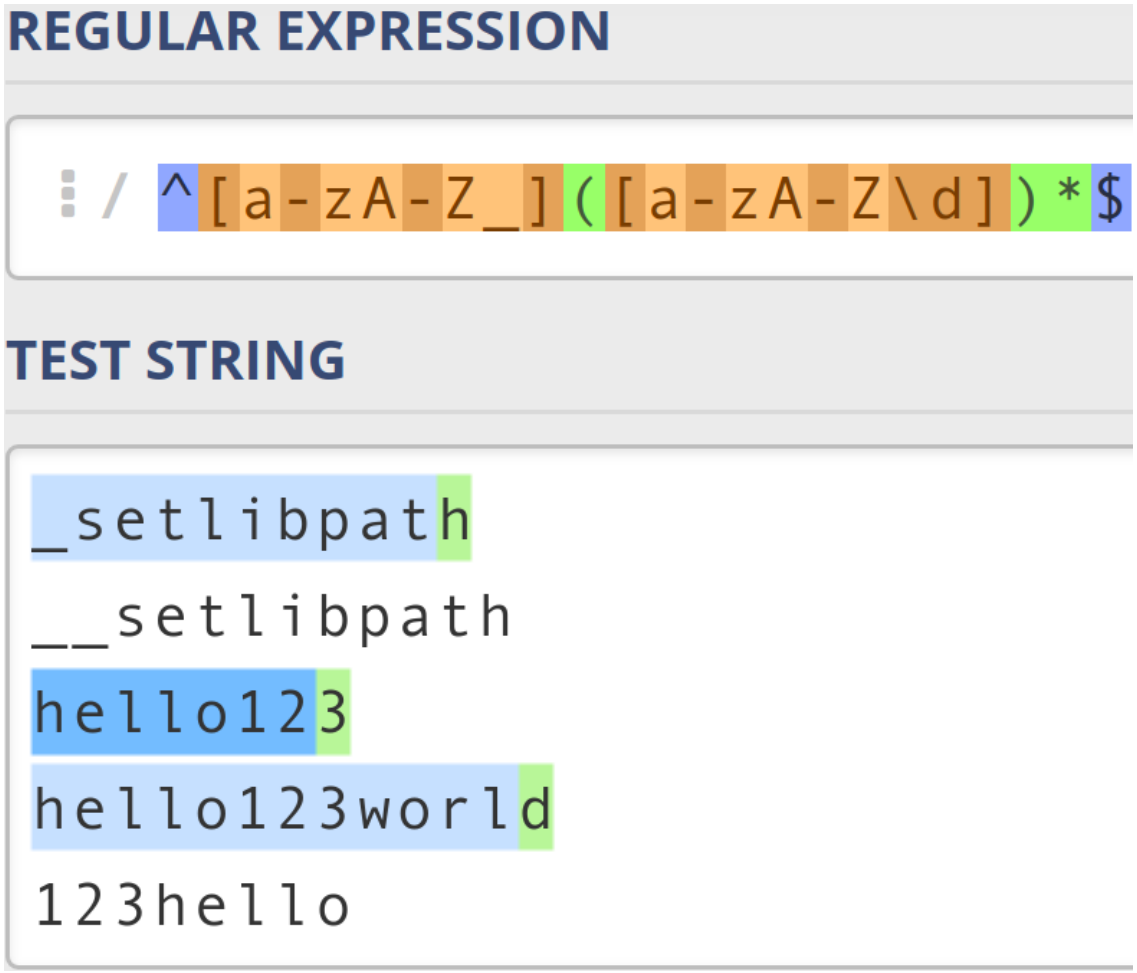
$[0-9]$ $[a-zA-Z]$ ^ $

正则定义与简记法

Vim	Java	ASCII	Description
	\p{ASCII}	[\x00-\x7F]	ASCII characters
	\p{Alnum}	[A-Za-z0-9]	Alphanumeric characters
\w	\w	[A-Za-z0-9_]	Alphanumeric characters plus "_"
\W	\W	[^A-Za-z0-9_]	Non-word characters
\a	\p{Alpha}	[A-Za-z]	Alphabetic characters
\s	\p{Blank}	[\t]	Space and tab
\< \>	\b	(?<=\W)(?=\w)|(?<=\w)(?=\W)	Word boundaries
	\B	(?<=\W)(?=\W)|(?<=\w)(?=\w)	Non-word boundaries
	\p{Cntrl}	[\x00-\x1F\x7F]	Control characters
\d	\p{Digit} or \d	[0-9]	Digits
\D	\D	[^0-9]	Non-digits
	\p{Graph}	[\x21-\x7E]	Visible characters
\l	\p{Lower}	[a-z]	Lowercase letters
\p	\p{Print}	[\x20-\x7E]	Visible characters and the space character
	\p{Punct}	[][!"#$%&'()*+,./:;<=>?@\^_`{|}~-]	Punctuation characters
_s	\p{Space} or \s	[\t\r\n\v\f]	Whitespace characters
\S	\S	[^ \t\r\n\v\f]	Non-whitespace characters
\u	\p{Upper}	[A-Z]	Uppercase letters
\x	\p{XDigit}	[A-Fa-f0-9]	Hexadecimal digits

C 语言中的标识符?

C 语言中的标识符?


REGULAR EXPRESSION
/ ^[a-zA-Z_]([a-zA-Z\d])*$
TEST STRING
_setlibpath
__setlibpath
hello123
hello123world
123hello

整数部分 . 小数部分 E/e 指数部分

整数部分 . 小数部分 E/e 指数部分

REGULAR EXPRESSION

6 matches, 225 steps (~1ms)

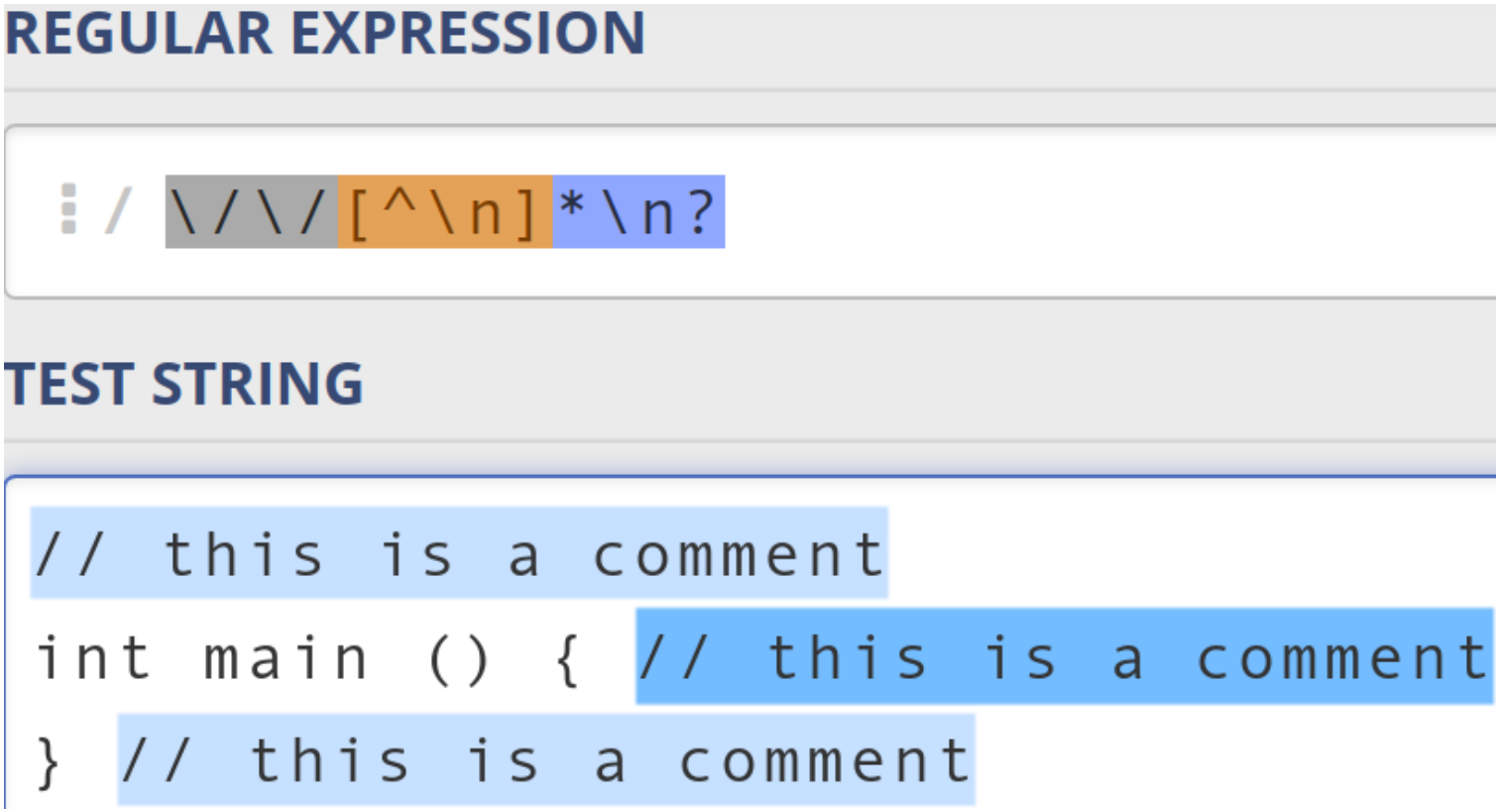
/ \d+(\.\d+)?([Ee][+-]?\d+)? / gm

TEST STRING

```
42 // The Answer to the Ultimate Question of Life, The Universe, and
Everything
2.99792458e8     // speed of light in vacuum
6.62606957E-34   // Planck constant
1.602176565e-19  // elementary charge
3.1415           // pi
2.718            // e
```

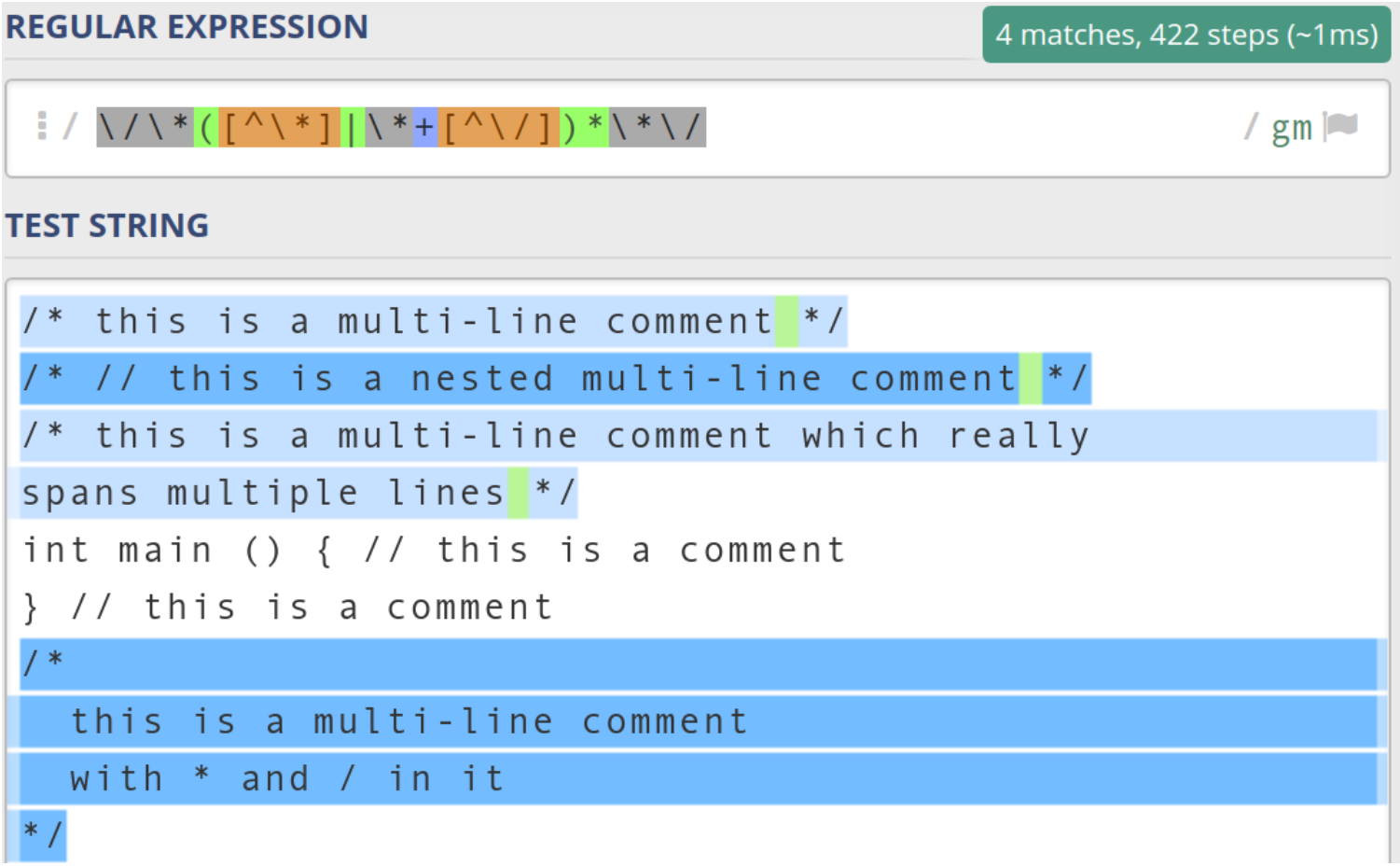
C 语言中**单行注释**对应的正则表达式?

C 语言中**单行注释**对应的正则表达式?



C 语言中**多行注释**对应的正则表达式?

C 语言中**多行注释**对应的正则表达式?



$$\Big(0|\big(1(01^*0)^*1\big)\Big)^*$$

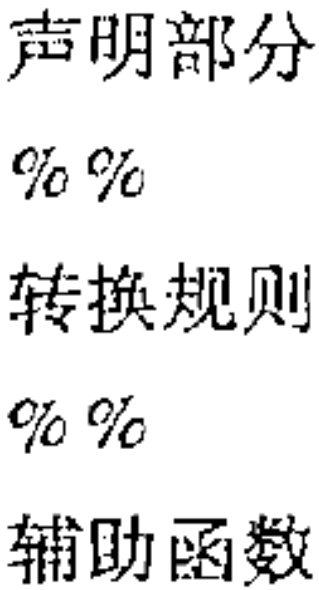
https://regex101.com/r/ED4qgC/1

Flex 程序的结构 (*.l* 文件)

声明部分: 直接拷贝到 *.c* 文件中

转换规则: **正则表达式 {动作}**

辅助函数: 动作中使用的辅助函数



```
%{
    /* 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 */
}
```

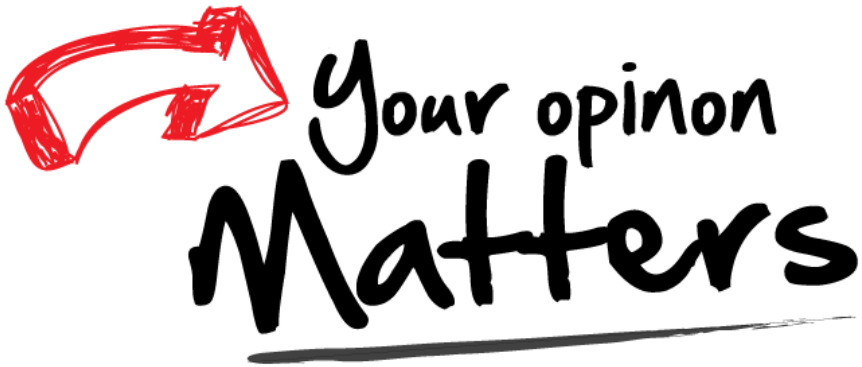
```
 8  /* fb1-1 just like unix wc (wordcount) */
 9  %{
10  int chars = 0;
11  int words = 0;
12  int lines = 0;
13  %}
14
15  %%
16
17  [^ \t\n\r\f\v]+   { words++; chars += strlen(yytext); }
18  \n                { chars++; lines++; }
19  .                 { chars++; }
20
21  %%
22
23  int main() {
24    yylex();
25    printf("%8d%8d%8d\n", lines, words, chars);
26  }
```

两大**冲突解决**规则

最前优先匹配：关键字

最长优先匹配：“>=”, “ifhappy”


Thank
You!



Office 926

hfwei@nju.edu.cn