Lex

Ing. Adrian Ulises Mercado Martínez

Enero 30, 2015

Lex

Lex

Lex

Lex

Lex es una de las herramientas que pueden ser utilizada para generar analizadores léxicos de manera automática, solo basta dar la descripción de las expresiones regulares que conforman el lenguaje.

Lex

2 Compilar con lex

Lex

Compilación con Lex y ejecución

lex ejemplo.l
gcc lex.yy.c -ll -o ejemplo
./ejemplo prueba.txt

Compilación con flex

flex ejemplo.l
gcc -o ejemplo lex.yy.c -lfl

Secuencia

- Lex crea un archivo fuente en C, llamado lex.yy.c.
- Lex.yy.c se compila para generar un ejecutable.
- El ejecutable analiza los archivos para producir los tokens.

Lex

3 Estructura de un programa en lex

Lex

Estructura de un programa Lex

```
Declaraciones

%%

Expresiones Regulares

%%

Procedimientos auxiliares

int main(){
    yylex();
    return 0;
}
```

Lex

Declaraciones

Esta sección del código contiene identificadores necesarios, expresiones regulares, constantes, los includes necesarios, etc.
Se puede incluir código en C usando:

```
%{
código en c
%}
```

Lex

Expresiones Regulares

Expresión Regular { Acción Léxica}

- Expresión Regular escrita en sintaxis de lex.
- Acción Léxica, escrita en código c.

Lex

Procedimientos Auxiliares

Son funciones que se utilizan para realizar determinados procesos necesarios para la obtención de los tokens, estos procedimientos dependen del lenguaje que se esté procesando.
Los procedimientos auxiliares se pueden llamar dentro de las acciones léxicas en la sección de expresiones regulares.

Lex

4 Expresiones Regulares

Lex

Metacaracteres

Expresión Regular	**Expresión Regular**
c	cualquier carácter c que no sea un operador
\c	el carácter c literalmente
"s"	La cadena s, literalmente
•	cualquier carácter excepto el salto de línea.
∧	inicio de línea
$	en fin de línea
[s]	cualquiera de los caracteres en la cadena s
[∧s]	cualquier carácter que no este en la cadena s

Lex

Metacaracteres

Expresión Regular	Expresión Regular
\n	nueva línea
r*	cero o mas instancias que coincidan con r
r+	una o más instancias que coincidan con r
r?	cero o una instancia de r
r{m,n}	entre m y n instancias que coincidan con r

Lex

Metacaracteres

Expresión Regular	Expresión Regular
$\mathbf{r_1 r_2}$	concatenación de r_1 con r_2
a | **b**	a o b
(r)+	una instancia de r
$\mathbf{r_1 / r_2}$	r_1 cuando va seguida de r_2

Lex

Ejemplos de Expresiones Regulares

Expresión Regular	Significado
abc*	ab, abc, abcc, abccc, ...
abc+	abc, abcc, abccc, abccccc, ...
a(bc)+	abc abcbc abcbcbc ...
a(bc)?	a abc
[abc]	a o b o c
[a-z]	cualquier caracter de **a** la **z**
[a\-z]	**a** o - o **z**
[-az]	- o **a** o **b**
[A-Za-z0-9]+	uno o más caracteres alfanuméricos.
[∧ab]	cualquier cadena excepto **a,b**
[a∧b]	**a** o ∧ o **b**
[a|b]	**a** o | o **b**
a|b	**a** o **b**

Lex

Ejemplos de Expresiones Regulares

Expresión Regular	Significado
*	El carácter asterisco
"**"	La cadena **
.*	Cualquier carácter que no se salto de línea cero o mas veces
abc$	Líneas que terminan con abc
∧**abc**	Líneas que empiezan con abc
a{1,5}	a, aa, aaa, aaaa, aaaaa
abc/123	

Lex

Ejemplos

Expresión Regular	Significado
\0	el ASCII de 0
\123	el ASCII de 123 en octal
\.	el ASCII de .
\x2A	el valor hexadecimal de 2A
r/s	r solo seguida de s
∧r	r solo al principio de línea
r$	r al final de la línea
<< *EOF* >>	el fin de archivo
r{2,5}	r de dos a cinco instancias

Lex

5 Funciones de lex

Lex

Funciones y variables de Lex

Nombre	Función
int yylex(**void**)	invoca al analizador y devuelve los tokens
char *yytext	apuntador al buffer
int yyleng	longitud de la cadena en el buffer
YYSTYPE yylval	valor asociado con el token
int yywrap(void)	retorna un 1 si termino y 0 si no puede terminar
FILE *yyout	archivo de salida
FILE *yyin	archivo de entrada
INITIAL	condición inicial
BEGIN	condición inicial de switch
ECHO	imprime la cadena que se le antepone

Lex

6 Retorno de tokens

Lex

Retorno de valores en Lex

Si se desea devolver un valor en lex se utiliza la palabra return, por ejemplo:

[0-9]+ {return NUMBER;}

En lex existen dos variables para el paso de valores ya sea hacia funciones escritas en C o a un parser diseñado con yacc. La variable yytext contiene la cadena de caracteres que son aceptados por la expresión regular. La variable yylval, definida como entera, se puede redefinir. Ejemplo:

[0-9]+ {yylval = atoi(yytext);
return NUMBER;
}

Lex

7 Ejemplos

Lex

```
digit [0-9]
letter [A-Za-z]
delim [ \t\n]
ws {delim}+
%%
{ws}{/* ignora espacios en blanco*/}
{digit}+ {printf("%s",yytext); printf(" = numero\n");}
{letter}{printf("%s",yytext); printf(" = palabra\n");}
%%
int main(){
yylex();
}
```

Lex

/* Scanner para un lenguaje estilo Pascal*/

```
%{
#include <math.h>
%}
 digito [0-9]
 id     [a-z][a.z0-9]*
 %%

{digito}  {
          printf("Un entero: %s (%d)\ n",yytext,
                  atoi(yytext ));
          }
{digito}+"."{digito}
```

Lex

```
/* Scanner para un lenguaje estilo Pascal*/

%{
#include <math.h>
%}
 digito [0-9]
 id     [a-z][a.z0-9]*
 %%

{digito}  {
          printf("Un entero: %s (%d)\ n",yytext,
                  atoi(yytext ));
          }
{digito}+"."{digito}
```

Lex

```
/* Scanner para un lenguaje estilo Pascal*/

%{
#include <math.h>
%}
 digito [0-9]
 id     [a-z][a.z0-9]*
 %%

{digito}  {
          printf("Un entero: %s (%d)\ n",yytext,
                  atoi(yytext ));
          }
{digito}+"."{digito}
```

Lex

```
/* Scanner para un lenguaje estilo Pascal*/

%{
#include <math.h>
%}
 digito [0-9]
 id     [a-z][a.z0-9]*
 %%

{digito}  {
          printf("Un entero: %s (%d)\ n",yytext,
                  atoi(yytext ));
          }
{digito}+"."{digito}
```

Lex

```
/* Scanner para un lenguaje estilo Pascal*/

%{
#include <math.h>
%}
 digito [0-9]
 id     [a-z][a.z0-9]*
 %%

{digito}  {
          printf("Un entero: %s (%d)\ n",yytext,
                  atoi(yytext ));
          }
{digito}+"."{digito}
```

Lex

```
/* Scanner para un lenguaje estilo Pascal*/

%{
#include <math.h>
%}
 digito [0-9]
 id     [a-z][a.z0-9]*
 %%

{digito}  {
          printf("Un entero: %s (%d)\ n",yytext,
                  atoi(yytext ));
          }
{digito}+"."{digito}
```

Lex

```
if|then|begin|end|procedure|function        {
            printf( "Palabra Reservada: %s\ n", yytext );
            }

{id}        {printf( "Identificador: %s\ n", yytext );}

"+"|"-"|"*"|"/"   { printf( "Operador: %s\ n", yytext );}

"{"[^] \ n]*"}"    {/* ignora comentarios */}

[ \ t\ n]+      {/* ignora los espacios en blanco */}

.           { printf( "Caracter no reconocido:
                %s\ n", yytext );}
```

Lex

```
%%

int main( int argc,char** argv )
{
    ++argv, --argc;  /* se salta el nombre del programa *
    if ( argc > 0 )
            yyin = fopen( argv[0], "r" );
    else
            yyin = stdin;

    yylex();
    return 0;
}
```