LEX

Estructura de un programa en LEX

{ definiciones }
%%
{ reglas }
%%
{ subrutinas del usuario }

Las definiciones y subrutinas son opcionales. El segundo %% es opcional pero el primer %% indica el comienzo de las reglas.

Expresiones Regulares

Las expresiones regulares en LEX son formadas en base a las siguientes reglas:

- Si se desea una expresión regular cuyo lenguaje que denota sea ***a*** entonces la expresión regular es **a**.

- Las operaciones básicas de una expresión regular (selección, concatenación, cerradura de Kleen y cerradura positiva, son representadas como :

  - **a | b** es la expresión regular que denota la selección entre **a** y **b**.

  - **ab** es la expresión regular que denota la concatenación de **a** y **b**.

  - **a*** es la expresión regular para indicar la cerradura de Kleen.

  - $\mathbf{a}^{+}$ es la expresión regular para indicar la cerradura positiva.

  - **Agrupamiento**. El agrupamiento es realizado con el uso de los operandos **(** y **)**.

LEX incluye otros operadores para darle mayor poder a las expresiones regulares y estos son:

- **Clases de Caracteres**. Son especificadas utilizando el operador **[ ]**, por ejemplo [abc], lo cual indica que se puede seleccionar entre el carácter **a**, **b** o **c**. Dentro de los corchetes, la mayoría de los operandos son ignorados, excepto:

- ****. Secuencia de Escape. Permite indicar números en octal, los cuales son considerados como caracteres, así como la indicación de considerar operadores como caracteres.

- **^**. Complemento del conjunto dentro de los corchetes, debe colocarse inmediatamente después del corchete izquierdo.

- **-**. Operador de rango, indica un rango de elementos dentro del conjunto, dicho rango se especifica en orden ascendente y su comportamiento está establecido por el código ASCII o EBCDIC.

  [a-z] [^a-z] [\0\n] [\0-\134]

- **Expresiones Opcionales**. El operador **?** indica que un elemento es opcional en una expresión.

  ab?c

- **Sensitivo al Contexto**. Son operadores para condicionar la aceptación de una cadena que cumpla con un cierto patrón.

  - **^**. Si el primer carácter de una expresión es **^**, entonces una cadena que cumpla con dicha expresión solo será aceptada si comienza al principio de una línea (después de una nueva línea y al comienzo de la entrada).

^abc

- **/**. Condiciona la aceptación de una cadena siempre y cuando termine con un cierto patrón.

  ab/cd

- **$**. Condiciona la aceptación de una cadena si esta termina con un carácter de nueva línea.

  ab$ = ab/\n

- **<>**. Indican que la aceptación solo re sealiza si se cumple con la condición que se encuentre dentro de los símbolos de mayor y menor que.

  <x> = ^x si la condición es que comience desde el principio de la línea

- **Repeticiones y Definiciones**. Los operadores **{** y **}**, sirven vara indicar repeticiones o para hacer uso de una constante previamente definida.

  {digito} Usa el valor de digito y sustituye el nombre por su valor en la expresión.
  a{1,5} Espera desde una a cinco ocurrencias de a.

- **Operadores para indicar operadores como caracteres.** Permiten tratar a los operadores como caracteres dentro de una expresión, dichos operadores son:

  - ****. Secuencia de Escape. Permite indicar números en octal, los cuales son considerados como caracteres, así como la indicación de considerar operadores como caracteres.

    \n \{ \%

  - **""**. Comillas dobles. Se coloca el carácter entre comillas y es tratado como tal y no como un operador, en caso de serlo.

    "\t" "[" "a"

- **Operador .**. Este operador sirve para aceptar cualquier otra cosa que no haya sido especificada en patrones previos. Si el punto es colocado como parte de una expresión permite indicar que se aceptan todos los caracteres excepto el fin de linea.

Acciones

Cuando una cadena que se está analizando cumple con una expresión, se ejecuta la acción asociada a dicha expresión. La acción que se asocia a la expresión se escribe en lenguaje C, pero LEX no realiza ninguna verificación sobre dicha acción, para LEX esta acción es como si fuera un comentario.

```
[ \t\n] ;
```

En esta regla, la acción es nula, por lo tanto, lo que se hace es que los caracteres dentro de los corchetes sean ignorados.

```
[a-z]+ printf( "%s", yytext );
```

En esta regla, la acción a realizar es la impresión del valor de yytext.
Si se requiere que una acción esté formada por más de una instrucción, entonces se usan los operadores **{** y **}**, donde las instrucciones son especificadas dentro de las llaves.

```
\"[^"]* { if (yytext[yyleng-1] == '\\')

    yymore();
    else
    ….
    }
```

Existen en LEX ciertas variables y funciones predefinidas, las cuales son utilizadas normalmente en la parte correspondiente a las acciones.
Cuando se requiere que dos o más reglas realicen la misma acción, se puede utilizar el operador | al final de cada expresión:

```
{d}{s}{e}{n} |
{d}{c}{o}{s} |
…
{d}{r}{e}{a}{l} printf( "%s"m yytext+1 );
```

Variables

- **yylex**. Variable de tipo char * que contiene la cadena que cumplió un patrón (lexema).
- **yyleng**. Variable de tipo int que contiene la longitud del valor contenido en yytext.

Funciones

- **yymore()**. Permite indicarle a LEX que la siguiente cadena que sea reconocida va al final de la cadena que se acaba de reconocer.
- **yyless(n)**. Permite indicarle a LEX que los n últimos caracteres en yytext son requeridos en este momento y deben ser reprocesados. Por ejemplo, si en C queremos resolver el problema de ambigüedad generado por "=-a" y suponiendo que se desea tratar como "=a" entonces:

```
=-[a-zA-Z] {

        printf( "Op ( =-) ambigüo\n");
        yyless( yyleng-1);
        …..

    }
```

Si se reguiere que sea tratado como "=-a", entonces tenemos

```
=-[a-zA-Z] {

        printf( "Op ( =-) ambiguo \n");
        yyless( yyleng-2);
        …..

    }
```

Una forma para escribir lo anterior sería:

=-/[A-Za-z] en el primer caso

=/-[A-Za-z] para el segundo

- **input()**. Regresa el siguiente carácter a ser leido.
- **output(c)**. Escribe el carácter c en el lugar indicado como salida.
- **unput(c)**. Coloca el carácter c en la entrada, para que posteriormente pueda ser leído.
- **yywrap()**. Es una función que regresa una valor de 1 para indicar que LEX termina el proceso de análisis al encontrar el fin de archivo, pero si regresa 0 al encontrarse el fin de archivo, esto le indica a LEX que el análisis debe continuar pero en un archivo diferente.
- **yylex()**. Función para inicial el análisis

LEX maneja especificaciones ambiguas, lo cual se presenta cuando una cadena de entrada cumple con más de un patrón, por lo que LEX realiza las siguientes acciones:

- Selecciona la regla en la que el empatamiento sea más largo.

- Para las reglas en donde el empatamiento es el mismo número de caracteres, se da preferencia a la primera regla en orden de aparición.

Definiciones

- Cualquier texto en un programa para LEX, que no es tratado, es copiado enteramente al programa generado.

- Cualquier línea que no sea parte de una regla en LEX o una acción que comience con un espacio en blanco o tabulador, son copiados al programa a ser generado.

- Cualquier cosa que haya sido incluida entre **%{** y **%}**. Los delimitadores son descartados. Normalmente son utilizados para indicar acciones de preprocesamiento (include, define, etc.).

- Cualquier cosa que aparezca después del segundo %%.

Las definiciones en LEX son especificadas antes del primer %%, donde cualquier línea en esta sección que no esté dentro de **%{** y **%}** y que comience en la columna uno, se asume que es una definición de sustitución de cadenas.

```
D [0-9]
E [DEde][-+]?{D}+
%%
{D}+ printf( "Entero" );
{D}+ "."{D}*({E})?|{D}*"."{D}+({E})?|{D}+{E}
```

Uso

- Compilar el programa escrito para LEX

  lex programa

- Compilar el programa generado por LEX con un compilador de C.

- Ejecutar el programa objeto resultante de la siguiente manera.

  programa >entrada <salida

LEX y YACC

LEX escribe una función llamada yylex(), la cual es requerida por YACC, para que pueda hacer su análisis.

Normalmente, la acción que se realiza cuando se cumple un patrón es regresar a YACC un token asociado con el patrón que se cumple.

```
return (token);
```

Donde los tokens son definidos en un archivo de encabezado y posteriormente es incluido

Ejemplo

```
/* Archivo de Encabezado */
#define identificador 0
#define numero 1
#define suma 2
#define error 3

{%
/* Archivo de LEX */
#include "encabeza.h"
%}
D [0-9]
L [A-Za-z]
%%
```

```
(_|{L})({L}|{D}|_)* return identificador;
{D}+ return numero;
+|\- return suma;
. return error;
%%

main()
{
yylex();
}
```

Sumario

El esquema general de un archivo fuente de LEX es

```
{definiciones}
%%
{reglas}
%%
{subrutinas}
```

La sección de definiciones esta formada por una combinación de

- Definiciones, en la forma **nombre espacio traslación**.
- Código, en la forma **espacio código**.
- Código, en la forma **%{ código %}**.
- Condiciones de inicio, en la forma **%S nombre1 nombre2 …**
- Tablas de conjuntos de caracteres, en la forma

  %T

  numero espacio cadena-de-caracteres

  %T

- Cambios de tamaños a arreglos internos

  %x nnn

  donde nnn es un número entero representando el tamaño de un arreglo y x puede ser cualquiera de los siguientes valores

  - ☐ p posiciones
  - ☐ n estados
  - ☐ e nodos del árbol
  - ☐ a transiciones
  - ☐ k clases empacadas de caracteres
  - ☐ o arreglo de salida.

Las líneas en la sección de reglas son de la forma **expresión acción**, donde la acción puede contener más de una línea usando las llaves como delimitadores.

Las expresiones regulares en LEX usan los siguientes operadores:

x el carácter "x".
"x" el carácter "x", aún si x es un operador.
\x el carácter "x", aún si x es un operador.
[xy] el carácter x o y.
[x-z] los caracteres x, y o z.
[^x] cualquier carácter excepto x.
. cualquier carácter excepto nueva línea.
^x una x al comienzo de una línea.
<y>x una x si se cumplió la condición y.
x$ una x al final de la línea.
x? Una x opcional.
x* 0, 1, 2, … ocurrencias de x.
x+ 1, 2, 3, … ocurrencias de x.
x|y una x o una y.
(x) una x.
x/y una x pero solo si es seguida por una y.
{xx} la sustitución de xx por su valor especificado en la sección de definiciones.
x{m,n} de m hasta n ocurrencias de x.