Universidad Nacional Autónoma de México

Facultad de Ingeniería

Práctica 2

Analizadores léxicos con Lex (flex)

ALUMNOS:

CÁRDENAS CÁRDENAS JORGE
MURRIETA VILLEGAS ALFONSO
REZA CHAVARRIA SERGIO GABRIEL
VALDESPINO MENDIETA JOAQUIN

PROFESOR:

ADRIAN ULISES MERCADO MARTINEZ

1. Objetivo

Que el alumno conozca y utilice los principios para generar analizadores léxicos utilizando lex.

2. Introducción

Lex es una herramienta para generar analizadores léxicos, que se deben describir mediante las expresiones regulares de los tokens que serán reconocidas por el analizador léxico (scanner o lexer).

Originalmente fue desarrollado para el sistema operativo Unix, pero con la popularidad de Linux se creo una versión para este sistema llamada flex.

Estructura de un archivo lex

Un programa en Lex consta de tres secciones:

```
Seccion de declaraciones
%%
Seccion de expresiones regulares
%%
Seccion de codigo de usuario (codigo en lenguaje C)
```

Sección de Declaraciones

Se utiliza para incluir los archivos de biblioteca y definir las variables globales.Su estructura es la siguiente:

```
%{
    #include <stdio.h>
    int contador;
}%
```

- **Macros.** Las macros son variables a las que se les asignar una expresión regular. Ejemplo letra [a-zA-Z], la macro se llama letra, separada por un espacio de la definición de la expresión regular.
- **Directivas.**Las directivas le indican a Lex que realice tareas extras al momento de generar el analizador léxico.
  1. %option yylineno genera un contador de lıneas automáticamente.
  2. %option noyywrap le indica a lex que debe generar de forma automática la función yywrap.
  3. %x o %s es para declarar estados léxicos.

Directivas de Código

- **Estructura:**

  Expresión Regular (espacio, nunca salto de lınea) Acción Léxica
- **Expresion Regular:**debe estar escrita con la sintaxis de Lex.
- **Acción Léxica:** se ejecuta cada vez que se encuentra una cadena que coincida con la expresión regular y es escrita en lenguaje C, encerrada por llaves.

Sección de código de usuario

En esta sección se escriben las funciones auxiliares para realizar el análisis léxico, por lo general es donde se agrega a main.

Carácter	Descripción
Argentina	Buenos Aires
c	Cualquier carácter representado por c que no sea un operador
\c	El carácter c literalmente
”S”	La cadena s, literalmente
.	Cualquier carácter excepto el salto de lınea
ˆ	Inicio de lınea
$	Fin de lınea
[s]	Cualquier carácter incluido dentro de la cadena s
[ˆs]	Cualquier carácter que no esté dentro de s
\n	salto de lınea
*	Cerradura de Kleene
+	Cerradura positiva
|	Disyunción
?	Cero o una instancia
{m,n}	entre m y n instancias

3. Desarrollo

1. Instalación de flex
   (a) Para Debian o Ubuntu: sudo apt-get install flex
   (b) Para Suse u OpenSuse: sudo zypper in flex
2. Para comprobar que se ha instalado correctamente:
   flex –version
3. Primer Programa en lex

```
%{
#include <stdio.h>
%}

digito [0-9]
letra [a-zA-Z]
espacio [\t\n}]
esps {espacio}

%%

{esps} {/*Ingnorar los espacios en blanco*/ }
{digito}+ {printf("Encontre un numero %s \n", yytext) ; }
{letra}+ {printf("Encontre una palabra %s \n" , yytext) ; }

%%

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

Pasos

(a) Transcribir el código anterior a un archivo con extensión .l, .lex o .flex.
(b) Compilar mediante la instrucción: flex archivo.l
(c) Comprobar se genero el archivo lex.yy.c

(d) Compilar mediante: gcc lex.yy.c -o nombreEjecutable -lfl

(e) Ejecutar

Preguntas

(a) ¿Qué ocurre si en la primera sección se quitan las llaves al nombre de la macro espacio?
-Lex identificaría literalmente la palabra espacio como una expresion regular.

(b) ¿Qué ocurre si en la segunda sección se quitan las llaves a las macros?
-De igual forma que en el caso anterior, lex identificaría tal cual las palabras como expresiones regulares.

(c) ¿Qué se guarda en yytext?
-La cadena reconocida por alguna expresion definida en lex.

(d) ¿Qué pasa al ejecutar el programa e introducir cadenas de caracteres y de dıgitos por la consola?
-Flex identifica las cadenas y caracteres por separado, es decir, muestra en consola una a una las subcadenas reconocidas a partir de las expresiones regulares digito y letra.

(e) ¿Qué ocurre si introducimos caracteres como "*" en la consola?
-Al no estar definido en ninguna regla, lex no lo asocia con algún comportamiento, de tal manera que solo muestra "*" en la consola.

4. Modificar al código anterior en un archivo nuevo, de tal manera que reconozca lo siguiente:

   (a) La expresión regular para los hexadecimales en lenguaje C

   (b) Los números decimales sin notación exponencial.

   (c) Los identificadores válidos del lenguaje C, con longitud máxima de 32 caracteres (Sugerencia: use el operador $\{m, n\}$).

   (d) Los espacios en blanco

   Compile y genere el ejecutable, realice varias pruebas para verificar que su programa funciona.

5. Escriba el siguiente codigo:

```
%{
#include <stdio.h>
%}

id [a-zA-Z]
espacio [\t\n}]+

%%
{id}+ {printf("Encontre un identificador %s \n", yytext) ; }
int {printf("Encontre una palabra reservada %s \n", yytext) ; }
float {printf("Encontre una palabra reservada %s \n", yytext) ; }
if {printf("Encontre una palabra reservada %s \n", yytext) ; }
else {printf("Encontre una palabra reservada %s \n", yytext) ; }

{espacio} {}

%%

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

Pasos

(a) Genere el ejecutable.

(b) Pruebe que el programa funciona introduciendo identificadores de solo letras.

(c) Pruebe el programa con las palabras reservadas descritas.

(d) Pase la expresión regular {id} al final de las palabras reservadas.

(e) Compile nuevamente el programa

(f) Vuelva a probar.

Preguntas

(a) ¿Qué pasa con las palabras reservadas?
-Lex las identifica segun corresponda (int, float, if, else).

(b) ¿Antes de hacer el cambio de la posición de {id} el programa se comporta de forma correcta con las palabras reservadas?
-No, Lex las consideraba como identificadores, dado que eran asociadas inicialmente dentro de esa categoria.

(c) ¿Qué sucede cuando se cambia el orden de la expresión regular {id}?
-Cambia el modo en que se identifica la cadena, dando así una prioridad a la primero que se declara.

(d) ¿Será importante la precedencia o prioridad de las expresiones regulares en lex?
-Sí, ya que lex reconoce cada cada cadena conforme lee las reglas, es decir, le da prioridad a lo primero que encuentra.

6. Manejo de errores

```
%{
#include <stdio.h>
%}

id [a-zA-Z]
espacio [\t\n}]+

%%
int {printf("Encontre una palabra reservada %s \n", yytext) ; }
float {printf("Encontre una palabra reservada %s \n", yytext) ; }
if {printf("Encontre una palabra reservada %s \n", yytext) ; }
else {printf("Encontre una palabra reservada %s \n", yytext) ; }
{id}+ {printf("Encontre un identificador %s \n", yytext) ; }x
{espacio} {/*Ignorar los espacios en blanco*/}
. {printf("Ha ocurrido un error lexico %s\n", yytext);}
%%

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

Pasos

(a) Genere el programa ejecutable

(b) Ejecute e introduzca sımbolos que su programa no debe reconocer

Preguntas

(a) ¿Qué pasa con los errores?
-Lex reconoce como error cualquier otra cadena que no corresponda con las expresiones regulares antes definidas, es decir, todo carácter declarado excepto el salto de linea.

(b) ¿Qué pasa si se cambia de posición la expresión regular ”.”?
-No se reconoceria ninguna regla definida despues de la expresión regular ”.”, dado ”.reconoce cualquier carácter.

4. Ejercicios

NOTA: Todos los ejercicios y sus respectivos archivos de salida y entrada se encontrarán en una carpeta comprimida y adjunta a la práctica.

5. Conclusiones

5.1. Cárdenas Cárdenas Jorge

Esta práctica me permitió comprender el funcionamiento a fondo de un analizador léxico, y su relación con las expresiones regulares y los autómatas finitos determinísticos, de tal manera que logre fortalecer los conceptos vistos en clase, todo esto con la ayuda del analizador léxico Flex.

Considero importante mencionar que el uso de Flex me permitió aclarar algunos de los conceptos de sintaxis y semántica del mismo, así como resaltar su importacia como analizador lexico dentro de un compilador.

5.2. Murrieta Villegas Alfonso

En la presente práctica se conoció y trabajó con Flex, que es un programa dedicado a la generación de analizadores léxicos.
A través del uso de conceptos previos como expresiones regulares, autómatas finitos y con las actividades dedicadas a conocer la sintaxis y estructura de los programas en flex, fue como se realizaron distintos ejercicios para poder no solamente tener todo lo visto en clase de forma teórica sino de forma tangible y visible, cabe destacar que en el caso de los autómatas se tuvieron que llevar a cabo a través de estados como forma de representación en código.
Por último, el uso de Flex como analizador léxico es más que importante debido a que es una herramienta que simplifica mucho todo el tratamiento léxico.

5.3. Reza Chavarria Sergio Gabriel

Con el uso de la herramienta de flex se pueden repasar los conceptos del análisis léxico junto con el repaso de las expresiones regulares y de los autómatas finitos. Flex ayuda a los códigos de C para tener implementación de los temas de expresiones para dar modificaciones al proceso de estos, dependiendo de las cadenas analizadas del léxico.

5.4. Valdespino Mendieta Joaquín

En conclusión en esta práctica se ha logrado observar la funcionalidad de lex para hacer analizadores léxicos mediante el uso de expresiones regulares en forma de código ,además de simular autómatas finitos mediante estados, esta herramienta nos facilita la tarea de reconocer los tokens de un texto o de un archivo de texto, por otro lado nos permite operarlos posteriormente, se comprendió y experimentó con la sintaxis de lex para realizar códigos relativamente más simples que cumplan con la tarea solicitada, por último se logró comprender lo que involucra la parte léxica de un lenguaje.

6. Referencias

Universidad de la República- Facultad de Ingeniería Instituto de Ingeniería eléctrica. El compilador GCC. Encontrado en: https://iie.fing.edu.uy/ vagonbar/gcc-make/gcc.htm